VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hledání optimální abecedy pro kompresi textových dat

Searching for Optimal Alphabet for Text Data Compression

2013 Bc. Petr Němec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Němec**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Hledání optimální abecedy pro kompresi textových dat
Searching for Optimal Alphabet for Text Data Compression

Zásady pro vypracování:

Cílem práce je nalézt optimální abecedu pro kompresi textových souborů daného typu. Vhodná abeceda obsahuje jen taková slova, která jsou opravdu potřebná a efektivně zvětšují kompresní poměr i při zvětšení velikosti abecedy.

Práce bude obsahovat:
1. Analýzu aktuálního stavu ve zpracovávané oblasti.
2. Návrh algoritmu pro řešení problému.
3. Implementace algoritmu.
4. Testování algoritmu.

Seznam doporučené odborné literatury:

[1] Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2013 .....

Rád bych poděkoval vedoucímu diplomové práce, Ing. Janu Platošovi, Ph.D. za vstřícnost při konzultacích, náměty, připomínky a pomoc při tvorbě této práce.

Poděkování patří také profesoru Ing. Ivanu Zelinkovi, Ph.D. za cenné rady a čas věnovaný konzultacím ohledně evolučních algoritmů.

Abstrakt

Hlavním cílem této práce je navrhnout a implementovat algoritmus pro hledání optimální abecedy pro kompresi dat v textové podobě. Je žádoucí nalézt takovou abecedu, která obsahuje jen nezbytně nutné symboly efektivně zvětšující kompresní poměr. Autor se zabývá spojením tří různých přístupů k textovým souborům. Pro tento optimalizační problém je vybrán samoorganizující se migrační algoritmus z třídy evolučních. Komprese je realizována metodou LZW se slovníkem reprezentovaným stromovou strukturou. Zhodnocení dosažených výsledků je provedeno na základě testů nad různými typy textových souborů. Celá práce je vytvářena v programovacím jazyce C#.

Klíčová slova: optimalizace, SOMA, evoluční algoritmy, komprese, abeceda

Abstract

Main goal of this work is to design and to implement optimal alphabet searching algorithm for text data compression. Is desirable to find alphabet contains only necessary symbols effectively increasing compression ratio. Author deals with the combination of three different approaches to text files. For this optimization problem is choosen selforganizing migration algorithm from evolutionary group. Compression is implement by LZW method, version with tree structured dictionary representation. Results evaluation is based on different types of text files testing. Whole work is created in programming language C#.

Keywords: optimization, SOMA, evolutionary algorithms, compression, alphabet

Seznam použitých zkratek a symbolů

SOMA	–	Samo-Organizující se Migrační Algoritmus
LZW	–	Lempel-Ziv-Welch
LZ77	–	Lempel-Ziv 77
LZ78	–	Lempel-Ziv 78
LZMA	–	Lempel-Ziv-Markov-Chain
PRT	–	Pertrubace
D	–	Dimenze
MP3	–	MPEG Audio Layer 3
RAR	–	Roshal ARchive
KP	–	Kompresní poměr

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Každý z nás se už jistě v životě setkal se situací, kdy musel vybírat při řešení určitého problému jednu z mnoha možných variant, o které si myslel, že je ta nejvýhodnější. Čili každý z nás se už jistě v životě setkal s optimalizací, i když ve velmi zjednodušené formě. Optimalizace je důležitou součástí života běžného člověka a v dnešním světě už také neodmyslitelnou součástí mnoha oblastí lidského výzkumu, jako např. v chemii, ekonomii, technice.

V této práci se autor zabývá optimalizací v oblasti informačních technologií. Optimalizace je zde implementována a využívána pro nalezení ideální množiny symbolů (znaků, slabik, slov), jenž se nazývá abeceda, pro kompresi dat v textové formě. Ona komprimace je další součástí této práce. Pro zhodnocení dosažených výsledků jsou provedeny testy nad množinou dat.

Druhá kapitola seznamuje s optimalizací, její historií a současným stavem. Jsou v ní popsány optimalizační algoritmy, princip funkce, využití. Další část je věnována algoritmům evolučním, jejich původu, významu a využití v různých odvětvích.

Ve třetí kapitole je rozebrán algoritmus, který byl vybrán pro optimalizaci problému obsaženého v této práci. A to algoritmus zvaný SOMA. Jeho původ, princip a chování, to vše obsahuje tato sekce.

V další kapitole autor popisuje praktický proces implementace optimalizační metody.

Pátá kapitola nabízí pohled na testovací funkce, nad kterými je prověřena funkčnost a přesnost implementovaného algoritmu SOMA.

Výsledky testování nad různorodými funkcemi popsanými v páté kapitole tvoří další část práce.

Přeformulování algoritmu SOMA z reálného do binárního prostoru uvádí sedmá kapitola.

V osmé části se čtenář seznámí s principem komprese dat. Budou nastíněny používané metody a pojmy z této oblasti.

Devátá část poukazuje na autorovu praktickou realizaci procesu hledání optimální abecedy. Jsou zde popsány všechny důležité kroky postupu.

Desátá kapitola obsahuje souhrn výsledku z testování nad různým typem textových dat.

V závěrečné kapitole jsou autorem zhodnoceny dosažené výsledky a zamýšlí se nad přínosem tvorby této práce.

2 Optimalizace

Už v antických dobách řešili lidé úlohy, u kterých se pokoušeli nalézt nejmenší či největší hodnotu pro daný problém. Byly to vesměs úlohy vznikající při praktické činnosti člověka. Velice známý je příběh, který pochází z devátého století př.n.l. Dcera tyrského krále Dido utekla od svého otce a chtěla vykoupit území v oblasti dnešního severu Afriky. Numidský král souhlasil s prodejem pozemku, avšak s podmínkou, že území nebude větší, než to které lze ohraničit kůží z dobytka. Dido dala volskou kůži rozřezat na velmi tenké proužky a po jejich svázání dostala provaz s určitou délkou, kterým maximalizovala rozsah své půdy. Tento příběh je pěkným příkladem optimalizační úlohy, ačkoliv pochází z let dávno minulých.

V dnešní době se setkáváme s řešením optimalizačních úloh v běžném životě, při řešení praktických záležitostí. Tyto úlohy mají například slovní zadání a řešíme je na základě našich znalostí, zkušeností a intuice. Tímto analytickým způsobem by však jistě nešly řešit úlohy složitějšího charakteru, jelikož by řešení bylo komplikované a velmi časově náročné. V takovém případě se k nalezení řešení využívá numerických postupů.

Optimalizaci můžeme popsat jako proces, u nějž se snažíme nalézt řešení, která jsou optimální, nebo se optimálnímu blíží. Obvykle nejsme schopni nalézt takové řešení v jednom kroku, ale je uskutečňován určitý algoritmus, během něhož dojdeme k řešení problému.

V první řadě by bylo dobré se zmínit o tom, co zmíněné optimální řešení představuje. Když budeme chtít optimalizovat jedno kritérium, tak optimum bude buď jeho minimum, nebo jeho maximum, v závislosti na tom, co v dané úloze hledáme. Jestliže vlastníme továrnu na výrobu chytrých telefonů, tak při dnešní poptávce se budeme zabývat především otázkou, jakým způsobem ***minimalizovat*** dobu potřebnou k jejich výrobě. Na druhé straně se určitě budeme snažit nakoupit materiál pro výrobu, platit zaměstnance a dělat reklamní kampaně tak, abychom ***maximalizovali*** náš zisk.

2.1 Optimalizační algoritmy

Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe a jejich využití se uvažuje v situacích, kde je řešení analytickým způsobem nereálné. Je tedy potřeba daný problém převést na problém, kde bude využito matematických zákonitostí a vztahů. Jinak řečeno je potřeba definovat problém matematickou funkcí. Taková matematická funkce se skládá z argumentů (neznámých), podle jejichž hodnot dostaneme výslednou hodnotu funkce, kterou optimalizujeme. Počet argumentů funkce je odvislý od daného problému a jeho složitosti.

V této kapitole je rozebrán problém globálních optimalizačních algoritmů, které se zásadně liší od klasických optimalizačních algoritmů, které se snaží nalézt alespoň jeden lokální extrém.

Definice 2.1 *Globální optimalizační algoritmy jsou optimalizační algoritmy, které zajišťují opatření, u kterého se předchází konvergenci k lokálním extrémům a zvyšuje pravděpodobnost nalezení globálních extrémů.*

2.1.1 Účelová funkce

Jak již bylo dříve zmíněno pro řešení určitého problému optimalizačními algoritmy je potřeba jej předefinovat tak, aby šel vyřešit numerickým výpočtem. Musíme tak nadefinovat funkci, která matematicky problém popíše. A právě optimalizace (nalezení minima nebo maxima) této funkce povede k hledaným optimálním hodnotám jejich argumentů. Pro takovou funkci se používá název účelová funkce, ale v mnoha literaturách se také můžeme setkat s výrazem fitness funkce, nebo cenová funkce.

2.1.2 Geometrická interpretace

U jakékoliv účelové funkce daného problému lze namodelovat její geometrickou interpretaci. Na řešení poté nahlížíme jako na hledání nejnižšího, či nejvyššího bodu v N-dimenzionálním prostoru, kde N je počet optimalizovaných parametrů fitness funkce. Snažíme se tedy najít globální extrémy funkce na dané hyperploše. Tím rozumějte, že máme najít všechny body, v nichž fitness funkce nabývá v prostoru svého maxima, a všechny body, v nichž funkce nabývá svého minima (samozřejmě, pokud tyto existují). Hledání (globálního) minima resp. maxima funkce je základní úlohou optimalizace v matematice. Nicméně je potřeba zmínit také pojem lokální extrém, neboli lokální minimum či maximum funkce. Některé algoritmy mají totiž tendenci konvergovat k nějakému takovému lokálnímu extrému a nakonec v něm uvíznout. Výsledné řešení pak není to nejvhodnější v daném prostoru možných.

Následující definice poukazují na matematickou interpretaci těchto pojmů.

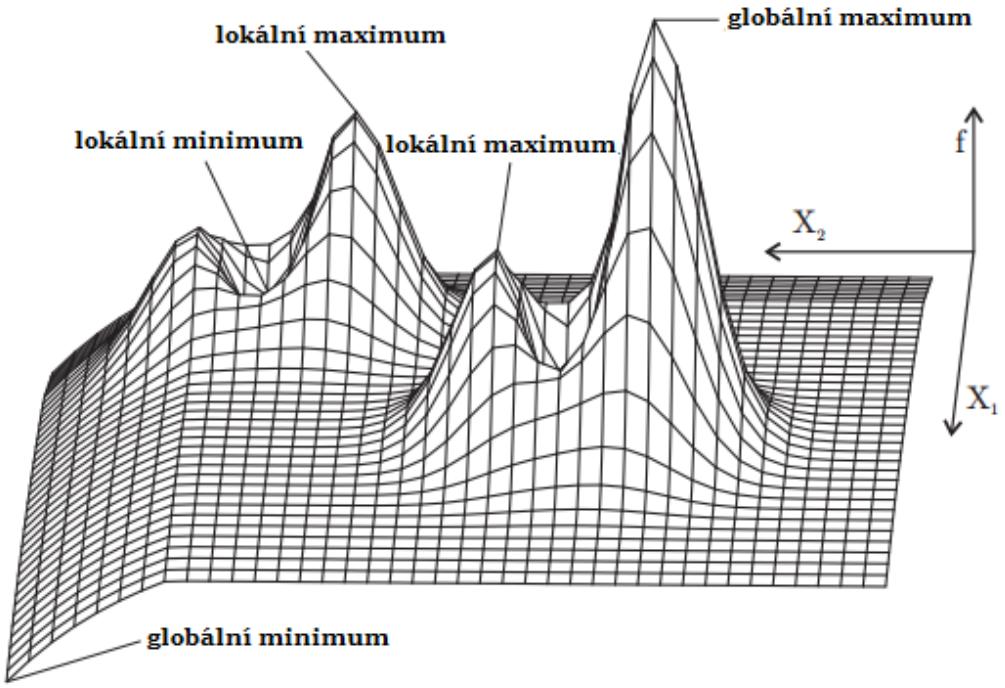
Definice 2.2 *Řekneme, že funkce* $f : \mathbb{X} \mapsto \mathbb{R}$ *má v bodě* $x_0 \in \mathbb{X}$ *lokální minimum, jestliže existuje ryzí okolí* $\overline{O}(x_0)$ *takové, že platí* $f(x_0) \leq f(x)$ *pro všechna* $x \in \overline{O}(x_0)$.

Definice 2.3 *Řekneme, že funkce* $f : \mathbb{X} \mapsto \mathbb{R}$ *má v bodě* $x_0 \in \mathbb{X}$ *lokální maximum, jestliže existuje ryzí okolí* $\overline{O}(x_0)$ *takové, že platí* $f(x_0) \geq f(x)$ *pro všechna* $x \in \overline{O}(x_0)$.

Definice 2.4 *Řekneme, že funkce* $f : \mathbb{X} \mapsto \mathbb{R}$ *má v bodě* $x_0 \in \mathbb{X}$ *globální minimum, jestliže platí* $f(x_0) \leq f(x)$ *pro všechna* $x \in \mathbb{X}$.

Definice 2.5 *Řekneme, že funkce* $f : \mathbb{X} \mapsto \mathbb{R}$ *má v bodě* $x_0 \in \mathbb{X}$ *globální maximum, jestliže platí* $f(x_0) \geq f(x)$ *pro všechna* $x \in \mathbb{X}$.

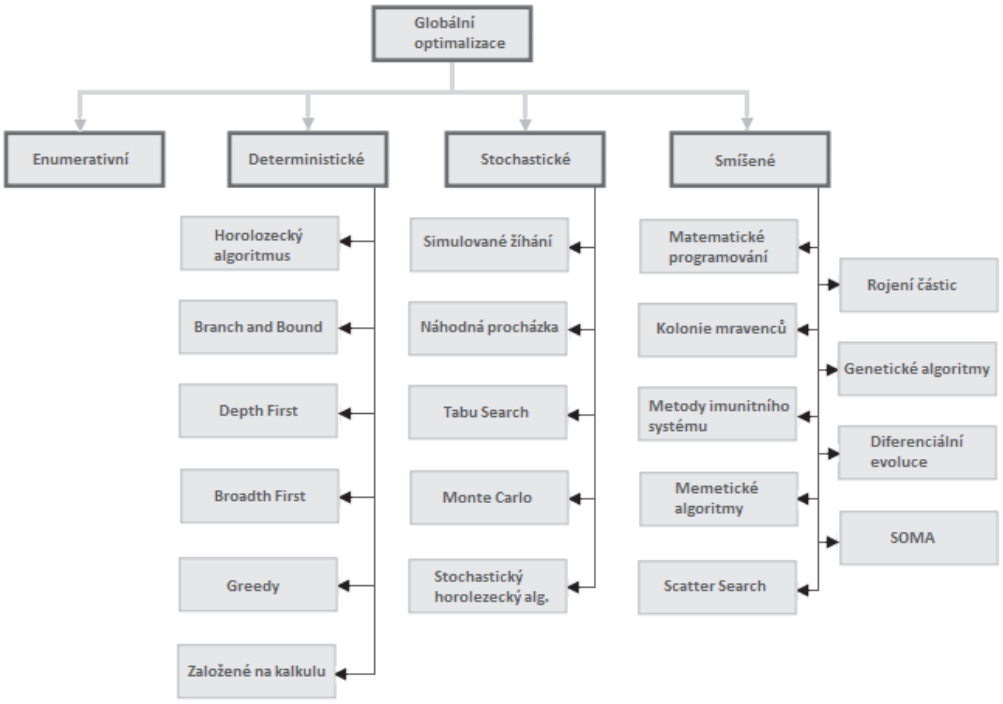
Obrázek 1 ilustruje náhodně vybranou funkci f definovanou ve dvoudimenzionálním prostoru $\mathbb{X} = (X_1, X_2)$. Je v něm nastíněn rozdíl mezi lokálním a globálním extrémem.



Obrázek 1: Globální a lokální optima dvou-dimenzionální funkce
[zdroj: `http://it-weise.de`]

Globální optimum je optimum celého prostoru možných řešení, zatímco optimum lokální ukazuje pouze na minimum či maximum daného subprostoru. Jak již bylo zmíněno, globální optimalizační algoritmy se snaží v prohledávaném prostoru najít extrémy globální. Proto se také používá výraz globální optimalizace problému.

Optimalizační algoritmy lze rozdělit podle různých faktorů - principu činnosti, složitosti atp. Jedno takové možné rozdělení, vyjadřující současný stav je zobrazeno níže.[7]



Obrázek 2: Rozdělení optimalizačních metod

2.2 Evoluční algoritmy

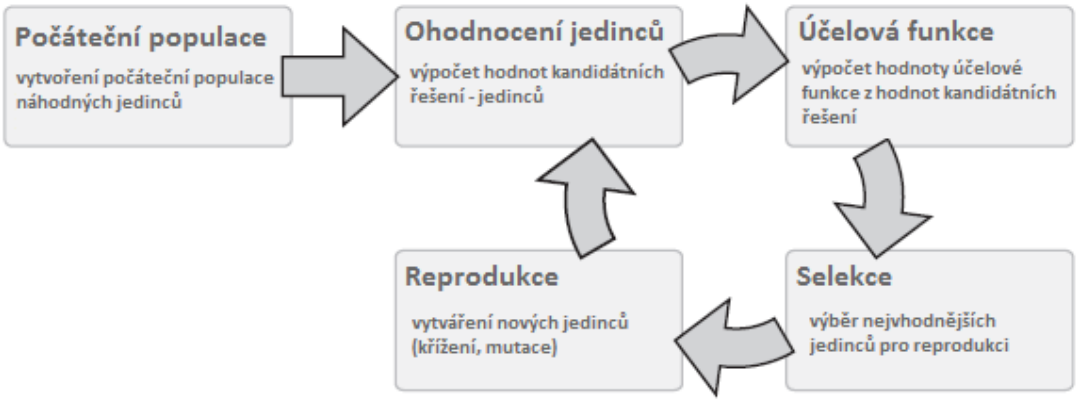
Slovo evoluce si jistě každý spojí s evolucí biologickou. Tu lze popsat, jako dlouhodobý a samovolný proces, během jehož průběhu se rozvíjí a rozčleňuje život na Zemi. Prakticky všichni vědci dnes vycházejí z Darwinovy evoluční teorie o postupném vývoji s přirozeným výběrem, tak jak ji Charles Darwin popsal v roce 1859 v jeho knize *On the Origin of Species*.

Evoluční algoritmy jsou metaheuristiky[1], které svým principem funkce napodobují biologický vývoj z přírody. Začaly se rozvíjet v polovině osmdesátých let a v dnešní době jsou už velmi sofistikované a začleněné do řešení vědeckých úloh.

[1]Stochastické vyhledávací metody

Jejich výhoda, v porovnání s ostatními optimalizačními metodami, tkví v jejich „black-box“ charakteru. Není totiž potřeba dokonale znát optimalizační algoritmy, stačí pouze správně formulovat účelovou funkci, která popíše řešenou problematiku. Tato třída algoritmů svým principem spadá do globální optimalizace a soustředí se tedy na nalezení extrémů globálních.

Cílem procesu evoluce je najít jedno nejlepší řešení z mnoha se nabízejících na základě práce s populacemi možných řešení, které aplikují princip přežití nejvhodnějšího.



Obrázek 3: Životní cyklus evolučních algoritmů

Na obrázku 3 je znázorněn základní princip běhu algoritmů založených na evoluci. Jejich účelem je cyklické vytváření nových populací a náhrada populací starých. To, jak budou jedinci v populaci reprezentováni a jaká pravidla budou plnit pak definuje typ evolučního algoritmu (genetické algoritmy, simulované žíhání, ...). Tím, že je neustále populace nahrazována jinou, tak se neustále v čase vyvíjí, a proto se používá označení „evoluční“.

Tato třída algoritmů nám pomáhá řešit velmi složité problémy efektivním způsobem a pro svou jednoduchost realizace získala oblibu v mnoha různých oborech. Například ve výzkumu umělé inteligence a teorie učení.

3 Samo-Organizující se Migrační Algoritmus

Samo-Organizující se Migrační Algoritmus, zkráceně SOMA, je algoritmus, který byl vyvinut v roce 1999. Základní myšlenka vedoucí k jeho vytvoření, spočívala v napodobení chování inteligentní skupiny jedinců, kteří spolupracují při hledání řešení společného problému. Tyto principy lze vypozorovat v přírodě jako např. hledání zdroje potravy. Jeho činnost je založena na geometrických principech a prací s populacemi, avšak na rozdíl od jiných evolučních algoritmů se v něm netvoří noví jedinci (potomci). Na základě spolupráce je zde prohledáván prostor možných řešení migrací oněch jedinců. Navzdory tomuto faktu je na něj nahlíženo jako na algoritmus spadající do třídy evolučních.

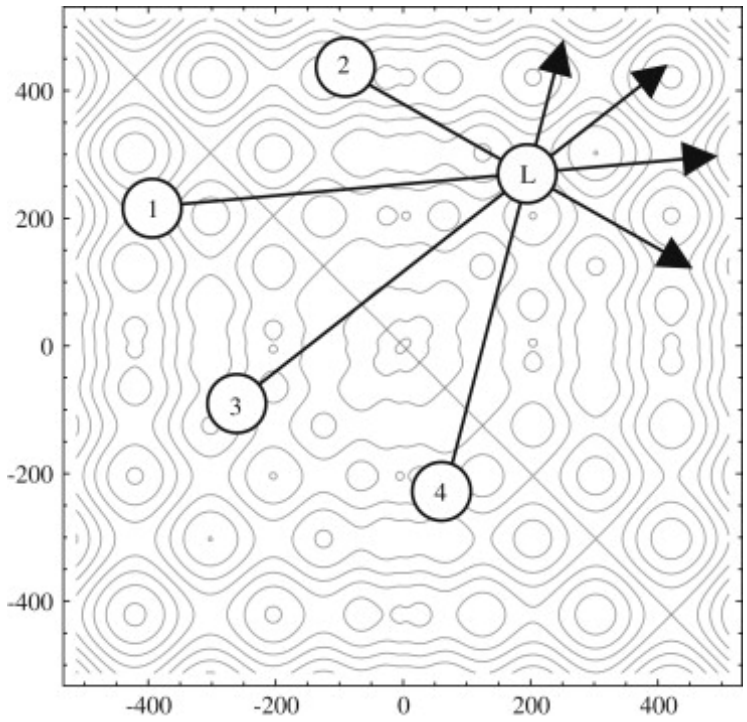
3.1 Princip

Princip soutěživě-kooperativní můžeme vypozorovat v přírodě, například u smečky vlků hledajících potravu. U takového hledání potravy jednotliví vlci spolu ve smečce spolupracují, ale v určitých chvílích dochází k rivalitě mezi nimi.

Když nastane fáze hledání, smečka se rozdělí a vlci mezi sebou **soutěží** v tom, který z nich se dostane k lepšímu zdroji obživy. Když jakýkoliv z nich něco objeví, **spolupracuje** a dá o tom vědět ostatním členům smečky. Každý z vlků se pak z různých míst začne přemisťovat k cílovému bodu a v případě, že cestou narazí na lepší zdroj potravy, opět to zahlásí celé smečce. Tyto fáze se neustále opakují, dokud není nalezena ta nejvýhodnější (nejkvalitnější) varianta. Princip je nastíněn na obrázku 4. Písmenem L je označen aktuální Leader a čísly 1 až 4 zbylí jedinci populace. Šipkami je znázorněn směr jejich migrace.

3.2 Populace

Populace, jak bylo zmíněno v sekci 2.2, a práce s ní je základním stavebním kamenem evolučních algoritmů. U SOMA jsou jedinci tvořeni na základě vzoru zvaného *Specimen*, jehož hodnoty určují hodnoty všech ostatních v populaci. Velikost populace, myšleno počet jejich členů, je udána jedním ze vstupních parametrů zvolených uživatelem viz 3.4. V principu evolučních algoritmů se populace během vývoje mění (stará je nahrazována novou). Tento děj však u SOMA ne-



Obrázek 4: Princip SOMA - verze All To One
[zdroj: www.sciencedirect.com/science/article/pii/S0895717711002998]

nastává. Prvopočáteční populace zůstává po celou dobu běhu optimalizace a u jedinců se mění jejich pozice v prostoru. Tímto faktem se SOMA úplně neřadí do evolučních metod[2].

3.3 Nalezení optima

Nalezení optimálního řešení problému je také zde zprostředkováno pomocí účelové funkce. Každému jedinci z populace je v běhu algoritmu spočítána a přiřazena jeho funkční hodnota, která se v čase mění. Je to ona vypovídající hodnota, jak kvalitní zdroj potravy je nalezen, pokud použijeme metaforu s vlky z úvodu této kapitoly. Jedinec mající nejvýhodnější funkční hodnotu, je ve verzi všichni k jednomu označen za vůdce.

[2]Spíše je na SOMA nahlíženo jako na algoritmus spadající do třídy memetických.

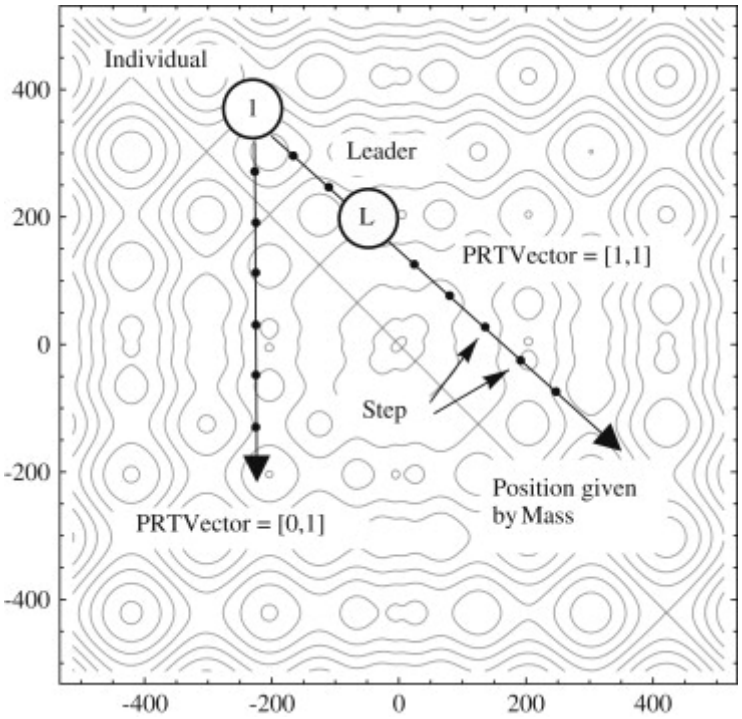
3.4 Parametry

Nadefinování hodnot parametrů je u všech optimalizačních algoritmů velmi důležitým aspektem. Nevhodně zvolené velikosti parametrů mohou mít za následek příliš dlouhou dobu běhu, nebo naopak bude algoritmus velmi svižný, avšak na úkor nalezení kvalitního řešení. Je tak nutné nalézt vyváženou kombinaci parametrů z hlediska času a kvality.

Parametry obvyklé u všech evolučních algoritmů*(D, NP, Migrace, AcceptedError)* jsou doplněny o další *(Mass, Step, PRT)*, které specifikují pohyb jedince k jinému po prostoru možných řešení. Jejich název je ponechán, jak je definoval autor algoritmu Ivan Zelinka ve své knize [1].

- **Mass** - relativní velikost přeskočení jedince, k němuž jiný jedinec migruje. Říkáme tím, v jaké vzdálenosti se má aktuálně migrující jedinec zastavit od vůdce.
- **Step** - v češtině „krok“, udává jak velkého přírůstku bude nabývat pozice migrujícího jedince, jinak řečeno jak velký krok v prostoru udělá aktuálně pohybující se jedinec směřující k Leaderovi.
- **PRT** - na základě tohoto parametru se generuje vektor hodnot zvaný pertrubační, který bude ovlivňovat, zda se daný jedinec bude pohybovat k Leaderovi přímo, či budou jeho kroky vychylovány. Tento parametr je, dá se říci, nejdůležitější, jelikož zavádí do algoritmu náhodnost. Díky tomu jsou prohledány ty části prostoru, kam by se určitý jedinec nikdy nemusel dostat.
- **D** - jednopísmenné označení tohoto parametru pochází od slova „Dimenze“. Jinak řečeno tímto parametrem definujeme počet optimalizovaných argumentů (parametrů) účelové funkce.
- **NP** - hodnotou tohoto parametru určujeme velikost počáteční populace. Kolik jedinců bude umístěno do prostoru hledaných řešení.
- **Migrace** - udává nám, kolikrát se populace přemístí do nových pozic, než bude běh algoritmu ukončen. Jiným názvem migrační kolo je ekvivalentem Generace z ostatních evolučních algoritmů.

- **AccpetedError** - jaký největší rozdíl může být mezi nejhorším a nejlepším jedincem v aktuální populaci je nastaveno tímto posledním parametrem. Hodnota rozdílu menší než hodnota tohoto parametru ukončuje běh algoritmu. Nastavení nulové, nebo záporné velikosti tento parametr eliminuje a proces optimalizace se zastaví až po dokončení všech migračních kol.



Obrázek 5: Grafické zobrazení parametrů Mass, Step a PRTVector
[zdroj: `www.sciencedirect.com/science/article/pii/S0895717711002998`]

3.5 Strategie

Jak již bylo řečeno, SOMA je algoritmus založen na principu soutěživě-kooperativním, kdy spolu jedinci v určitý čas soutěží a po jinou dobu nastává fáze spolupráce. Z tohoto důvodu jsou jednotlivé varianty SOMA nazývány strategiemi.

3.5.1 All To One

Strategie anglicky pojemovaná All To One lze přeložit jako *Všichni k jednomu*. Jak už název napovídá, tak zde každý jedinec z populace migruje k jedinému vůdci (Leadrovi). Tento postup je opakován v každém migračním kole a Leaderova pozice je v jeho rámci neměnná.

3.5.2 All To All

All To All, v překladu strategie *všichni ke všem* nemá daného Leadera. Každý jedinec migruje k jinému jedinci. Celá populace se přemístí do nových pozic po skončení jednoho migračního kola, tzn. po ukončení putování všech jedinců ke všem zbývajícím.

3.5.3 All To All Adaptive

Adaptivně všichni ke všem je třetí strategie SOMA. Princip je obdobný jako u strategie All To All, tedy vůdcem se stává postupně každý z jedinců. Rozdíl je zde následující: aktuálně migrující jedinec se přesouvá do nové pozice bezprostředně po každé zrovna dokončené migraci ke každému jedinci z populace. Z této nové pozice poté migruje ke všem zbývajícím jedincům.

3.5.4 All To All Rand

Tato strategie opět využívá funkce vůdce. Ten ale není určen nejlepší pozicí na hyperploše, tak jak tomu bylo u první z variant algoritmu, ale je dán náhodným výběrem z populace.

3.5.5 Clusters

Jedinci jsou zde rozděleni do tzv. svazků (Clusters). V každém takovém svazku probíhá SOMA algoritmus. Je tak možno použít jakoukoliv z předchozích strategií pro jednotlivé Clustery.

4 Implementace SOMA

Celý princip a vlastnosti SOMA v kapitole 3 autor realizuje jako softwarovou aplikaci, jenž bude posléze využívána pro řešení další, praktické části diplomové práce. Zdrojový kód byl tvořen ve vysokoúrovňovém programovacím jazyce *C#*[3] za použití vývojového prostředí *Visual Studio 2012*.

Pro implementaci byla vybrána strategie SOMA algoritmu **ALL TO ONE** popsaná v kapitole 3.5. Byla otestována také varianta **ALL TO ALL**, jejíž princip umožňuje nalezení řešení, velmi se blížící optimálnímu. Výpočetní náročnost se však ukázala být mnohonásobně vyšší, což se samozřejmě projeví v celkovém čase, potřebném k řešení. Jelikož je proces hledání optimální abecedy velmi složitý (pohyb ve více než 500 dimenzích), nebyla tato varianta z časové náročnosti vybrána.

Naimplementovaný SOMA bude v další části práce otestován vybranými funkcemi, na kterých se zjistí, zda pracuje správně a jak přesný, z hlediska hodnoty globálního extrému, dokáže být.

[3] v rámci .NET Framework 4.5

Celý proces samoorganizujícího se migračního algoritmu ve vybrané strategii lze ve velmi zjednodušené podobě popsat následujícím pseudokódem.

```
procedure SOMA_AllToOne()
begin
    nacti ridici a ukoncovaci parametry
    vytvor Specimen
    vytvor populaci
    ohodnot populaci
    for i := 1 to pocet migracnich kol do
        vyber z populace Ledera
        generuj PRT Vektor
        for each jedinec j in populace do
            for s := 1 to pocet kroku do
                jedinec j migruj k Leaderovi
                spocti hodnotu ucelove funkce
                if nova ucelova hodnota jedince j > ucelova hodnota jedince j then
                    ucelova hodnota jedince j := nova ucelova hodnota jedince j
            end
        end
        presun jedince j na nejlepsi pozici
    end
end
```

Výpis 1: Pseudokód běhu SOMA All To One

Implementace algoritmu byla vytvořena v třídě s názvem *Soma_AllToOne*. Zde se nacházejí všechny důležité funkce zajišťující běh této optimalizační metody.

Další třídy byly vytvořeny pro přehlednější práci s algoritmem.

- *Parameters* - třída, v níž se nacházejí všechny vstupní parametry algoritmu.
- *Specimen* - reprezentuje vzor pro tvorbu jedinců. Nastavení hranic dimenzí se děje právě zde.
- *Individual* - třída popisující vlastnosti jedince. Obsahuje jeho souřadnice v prostoru a aktuální fitness hodnotu.
- *Population* - zastřešuje množinu jedinců tvořící populaci.

4.1 Definice parametrů

Všechny parametry popsané v kapitole 3.4 jsou uživatelem definované a musí být zvoleny před započetím samotného běhu algoritmu. Pro jednoduchost je množina parametrů uložena v textovém souboru spolu s dalšími konfiguračními hodnotami, které budou popsány dále. Každý parametr musí být vepsán na řádek v určeném pořadí a na tomto řádku se nesmí nacházet žádné další znaky (řetězce).

Definované parametry jsou buď celočíselné (NP, D, Migrations), nebo pochází z množiny reálných hodnot (Mass, Step, PRT, AcceptedError).

Na výpisu kódu 2 je ukázáno načtení v jazyce C#. Po krocích se postupně čtou řádky vstupního souboru s parametry, které se ukládají do vnitřních proměnných.

```
private void LoadParameters(StreamReader parametersFile)
{
    parameters = new Parameters();

    try
    {
        double.TryParse(parametersFile.ReadLine(), System.Globalization.NumberStyles.Any,
            CultureInfo.InvariantCulture, out parameters.Mass);
        double.TryParse(parametersFile.ReadLine(), System.Globalization.NumberStyles.Any,
            CultureInfo.InvariantCulture, out parameters.Step);
        double.TryParse(parametersFile.ReadLine(), System.Globalization.NumberStyles.Any,
            CultureInfo.InvariantCulture, out parameters.PRT);
        int.TryParse(parametersFile.ReadLine(), out parameters.NP);
        int.TryParse(parametersFile.ReadLine(), out parameters.D);
        int.TryParse(parametersFile.ReadLine(), out parameters.Migrations);
        double.TryParse(parametersFile.ReadLine(), System.Globalization.NumberStyles.Any,
            CultureInfo.InvariantCulture, out parameters.AcceptedError);
    }
    catch (Exception)
    {
        Console.WriteLine("Parameters loading failed!");
    }
}
```

Výpis 2: Načtení řídících a ukončovacích parametrů

Tabulka níže poukazuje na povolené rozsahy hodnot pro parametry.

Parametr	Rozsah hodnot
Mass	$\langle 1.1, 5 \rangle$
Step	$\langle 0.11, Mass \rangle$
PRT	$\langle 0, 1 \rangle$
D	dáno problémem
NP	$\langle 2,$ dáno uživatelem$\rangle$
Migrace	$\langle 10,$ dáno uživatelem$\rangle$
AcceptedError	libovolně definuje uživatel

Tabulka 1: Rozsah hodnot parametrů SOMA

4.2 Tvorba populace

Po načtení parametrů následuje vytvoření jedinců reprezentujících populaci prostředí možných řešení. Programově se tak děje pomocí definovaného vzoru *Specimen*, který je vytvořen před samotnou inicializací počáteční populace z parametrů, jež definuje uživatel. Do souboru zadává pro každý parametr Specimenu trojici. První dva údaje vymezují interval, ve kterém se může jedinec pohybovat. Třetí určuje zda jsou hodnoty celočíselného, nebo reálného typu. Počáteční populace je vytvořena náhodně. Jinými slovy řečeno je nový jedinec umístěn do prostoru na základě náhodné hodnoty, která se však musí pohybovat v rozmezí zmíněného Specimenu.

Každý jedinec je tvořen množinou hodnot, udávající jeho pozici v prostoru. Pokud tedy budeme řešit problém, u něhož je potřeba optimalizovat hodnoty osmi parametrů, čili budeme se pohybovat v 8D prostoru, bude kardinalita této množiny právě 8. Zároveň je každému individuu ještě vypočítána jeho Fitness hodnota, která se v každém migračním kole může změnit.

V tabulce 2 je demonstrováno, jak jsou reprezentováni jedinci. Každý řádek obsahuje hodnoty jedince označené P1-P8, čtěte parametr1-parametr8, které reprezentují jeho pozici na hyperploše. V posledním sloupci jsou uváděny jejich účelové hodnoty, které určují směr evoluce.

	P1	P2	P3	P4	P5	P6	P7	P8	**Fitness hodnota**
Jedinec1	0.18	98.00	76.31	4,56	19.09	73.11	14,40	16,60	50.13
Jedinec2	1.90	87.89	77.80	7.43	30.90	101.78	34.21	16.19	90.23
Jedinec3	2.01	89.12	49.09	1.67	29.56	98.01	33.45	16.99	91.91
Jedinec4	0.45	103.74	80.39	1.13	30.18	88.91	19.56	12.34	32.11
⋮									
JedinecNP	0.99	56.56	70.01	5.67	41.34	95.55	20.02	15.98	22.22

Tabulka 2: Reprezentace populace

Metoda inicializace populace, neboli její prvotní vytvoření z náhodných hodnot v C#. V cyklu je procházena celá populace, jedincům jsou přiřazovány hodnoty a následně je spočítána hodnota účelové funkce.

```
private void InitializePopulation ()
{
    population = new Population(parameters.NP);
    Random random = new Random();

    for (int i = 0; i < parameters.NP; i++)
    {
        population[i] = new Individual(parameters.D);

        for (int j = 0; j < parameters.D; j++)
        {
            population[i][j] = random.NextDouble() * (specimen[j].max - specimen[j].min) +
                specimen[j].min;
        }
        getFitnessValue(population[i]) ;
    }
}
```

Výpis 3: Inicializace populace

4.3 Pertrubační vektor

Zkráceně PRTVektor ovlivňuje pohyb jedince na základě uživatelem zvoleného PRT parametru. Pro každý parametr z D je dílem náhody vektor plněn hodnotami 1 nebo 0 podle toho, zda vygenerované reálné číslo je větší nebo menší než PRT parametr.

```
for (int i = 0; i < parameters.D; i++)
{
   if (random.NextDouble() > parameters.PRT)
      PRTVector[i] = 0;
   else PRTVector[i] = 1;
}
```

Výpis 4: Generace pertrubačního vektoru

Když bude hodnota pro určitý parametr rovná 0, tak se jedinec nebude v dané dimenzi pohybovat, jelikož se nulou vynásobí jeho plánovaná délka skoku.

4.4 Účelová funkce

Fitness funkce popisující řešenou problematiku je ve zdrojovém kódu definována matematickým vztahem. Pro výpočet je jako parametr metody předán jedinec, jehož souřadnice v prostoru představují hodnoty jednotlivých argumentů. Po dosazení do rovnice dostaneme skalár, jenž nám říká, jak vhodný je daný jedinec pro další vývoj populace.

4.5 Migrační kola

Algoritmus SOMA pracuje v cyklech zvaných Migrační kola. Jedno takové migrační kolo plní stejný význam jako Generace u genetických algoritmů. Během migračního kola však nejsou vytvářeni noví jedinci. Počáteční populace pouze migruje skrz prostor možných řešení. Dá se říci, že jedinci jsou přesouváni do určité pozice pomocí skoků směrem k jinému jedinci.

4.6 Zpráva o výsledcích

Jakmile je proces optimalizace dokončen, tak je uživateli do konzole zobrazen výstup. Ten obsahuje informace o dosažených výsledcích.

Uživateli budou zobrazeny následující údaje:

- **Fitness hodnota nejlepšího** z populace, to znamená hodnotu účelové funkce od jedince, jehož kombinace parametrů je pro hledaný problém optimální.
- **Fitness hodnota nejhoršího** z populace, to znamená hodnotu účelové funkce od jedince, jehož kombinace parametrů je pro hledaný problém nejméně vhodná.
- **Počet migračních kol**, které bylo třeba v rámci algoritmu vykonat pro nalezení optimálního řešení viz 4.5.
- **Počet ohodnocení účelové funkce**. Pro představu kolikrát byla počítána účelová funkce pro jedince v populaci slouží tento údaj.
- **Nalezená optimální sestava všech parametrů**. Nejdůležitější výstup pro uživatele. Jsou vypsány všechny parametry a k nim je přiřazena hodnota každého z nich. Všechny tyto hodnoty zajistily optimální kombinaci pro řešení problému.
- **Celkový čas** potřebný k nalezení globálního extrému.
- **Nastavené vstupní parametry.** Hodnoty uživatelem zvolených parametrů.

Celý tento výpis je také uložen do souboru. Do souboru se také ukládá, jak se vyvíjelo ohodnocení účelové funkce v čase. Jinými slovy tato historie zobrazuje, jaké nejlepší řešení bylo v jednotlivých migračních kolech nalezeno.

5 Testovací funkce

V této kapitole je možné nahlédnout na testovací funkce, nad kterými jsou prováděné testy SOMA algoritmem. Na základě těchto funkcí lze porovnávat optimalizační algoritmy. Pojem testovací funkce označuje sadu speciálních účelových funkcí, u kterých jsou hledány jejich globální extrémy. Tedy globální minimum, či globální maximum. Jelikož mají tyto funkce velmi specifický průběh (jsou nelineární, multimodální atd.), dokážeme tímto způsobem ověřit chování optimalizačních algoritmů. Nejdůležitějším faktem je to, že u většiny z těchto testovacích funkcí dopředu známe hodnoty jejich extrémů (zpravidla globální minimum) pro libovolný počet parametrů. Tím je nám umožněno exaktně porovnat, zda funkční hodnota globálního extrému nalezeného testovaným optimalizačním algoritmem se blíží k hodnotě definované.

Funkcí pro testování je nepřeberné množství. V tomto případě jich bylo vybráno 13, od těch jednodušších, až po ty velmi složité. Komplikovanost jednotlivé funkce je možno posoudit podle jejího průběhu v trojrozměrném grafu, popřípadě podle přiložených předpisů jejich funkcí. Tyto předpisy slouží jako fitness funkce a na základě její hodnoty se hledá globální extrém.

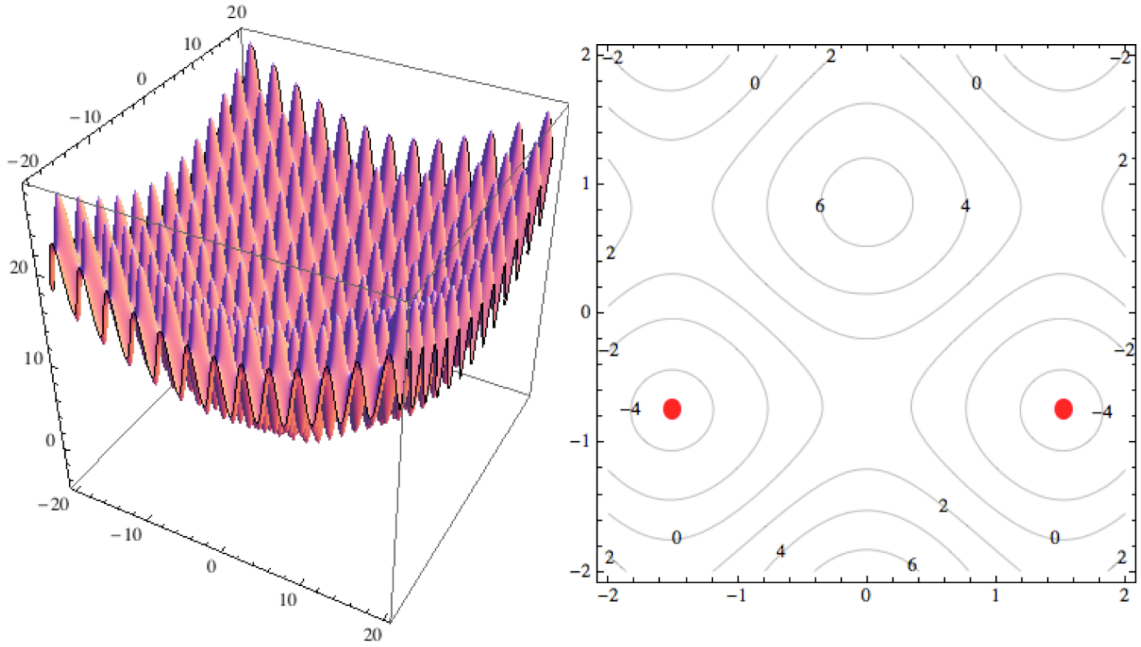
Grafy funkcí jsou vyexportovány z programu *Mathematica*, kde byly namodelovány[10].

5.1 Ackleyho funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1} \left(\frac{1}{e^5} \sqrt{\left(x_i^2 + x_{i+1}^2\right)} + 3\left(\cos\left(2x_i\right) + \sin\left(2x_{i+1}\right)\right) \right)$$

Průběh funkce:



Obrázek 6: Grafy Ackleyho funkce
[červené body vpravo označují pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = -7,54276 - 2,91867 * (n-3)$

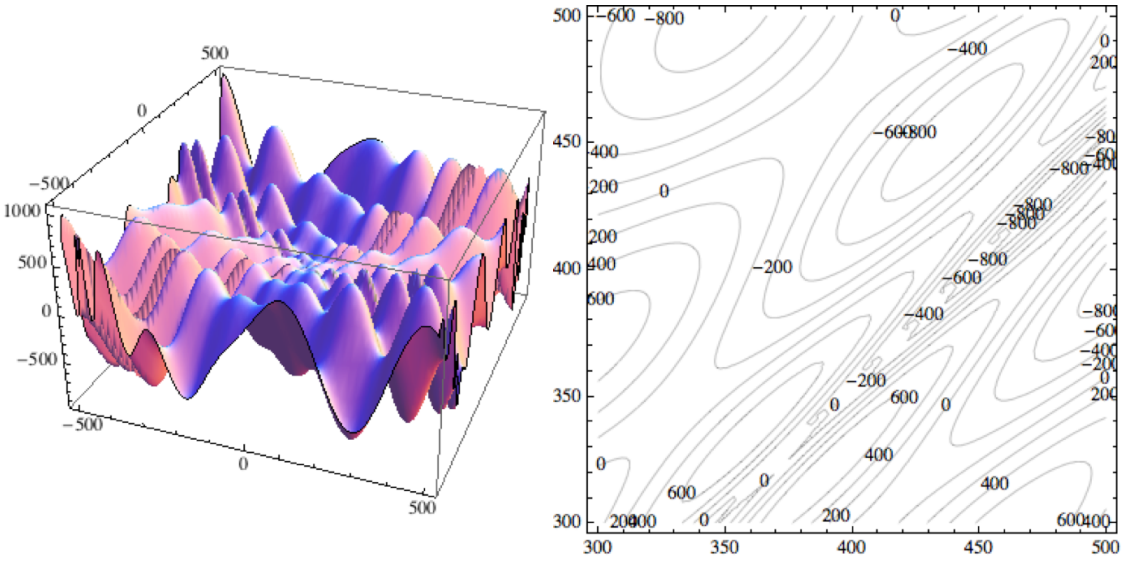
kde n je počet dimenzí prostoru.

5.2 Plato vajec

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1} \left(-x_i \sin(\sqrt{|x_i - x_{i+1} - 47|}) - (x_{i+1} + 47) \sin(\sqrt{\left|x_{i+1} + 47 + \frac{x_i}{2}\right|}) \right)$$

Průběh funkce:



Obrázek 7: Grafy funkce Plato vajec

[zdroj: `http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

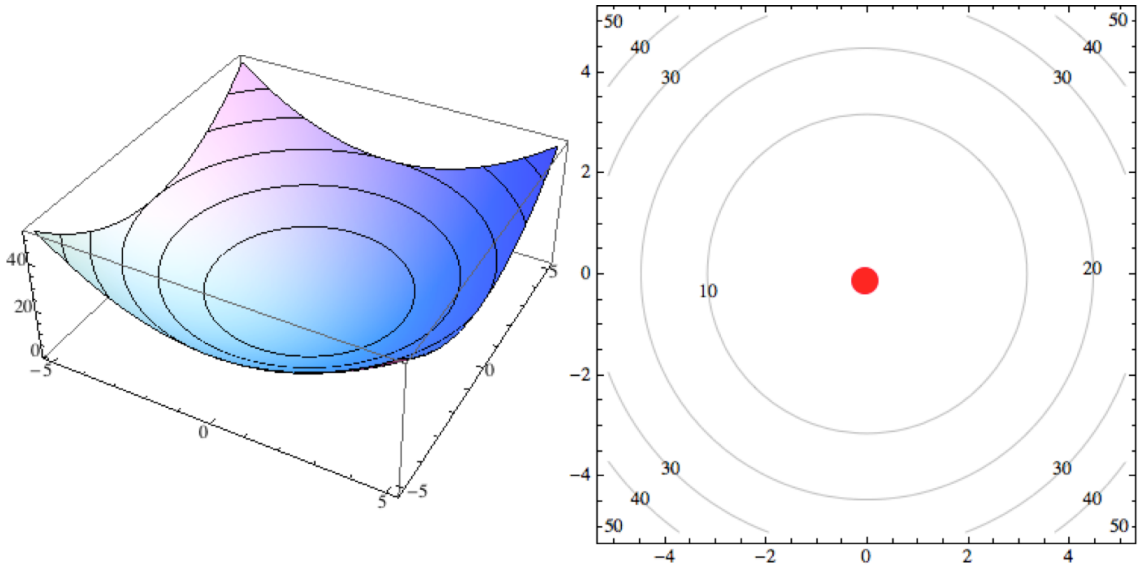
Přesná hodnota globálního minima nebyla v žádné literatuře zjištěna.

5.3 První de Jongova funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D} x_i^2$$

Průběh funkce:



Obrázek 8: Grafy První de Jongovy funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

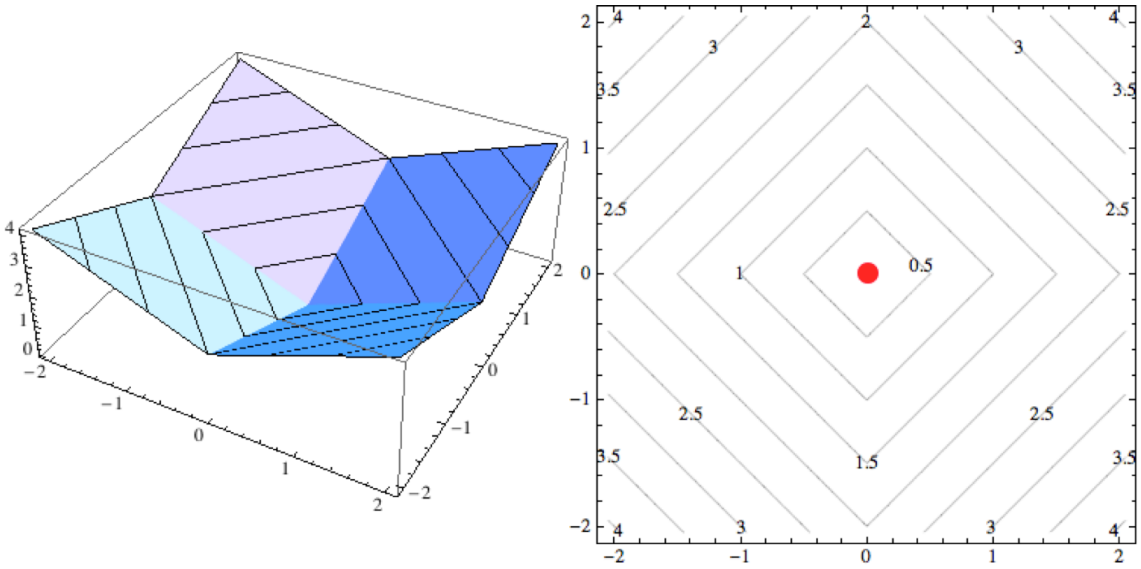
$f(x) = 0$

5.4 Třetí de Jongova funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D} |x_i|$$

Průběh funkce:



Obrázek 9: Grafy Třetí de Jongovy funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

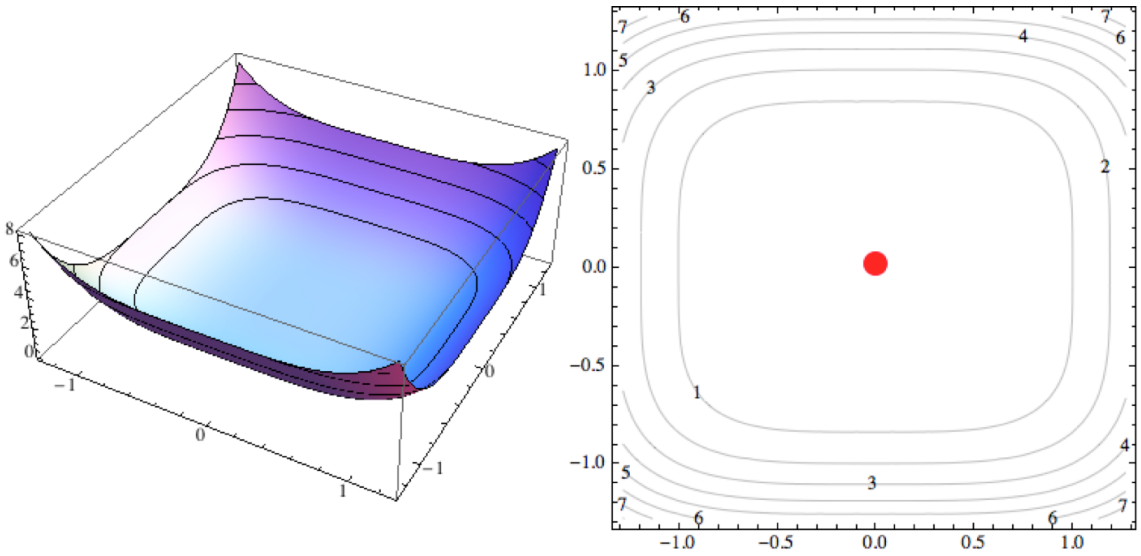
$f(x) = 0$

5.5 Čtvrtá de Jongova funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D} i x_i^4$$

Průběh funkce:



Obrázek 10: Grafy Čtvrté de Jongovy funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

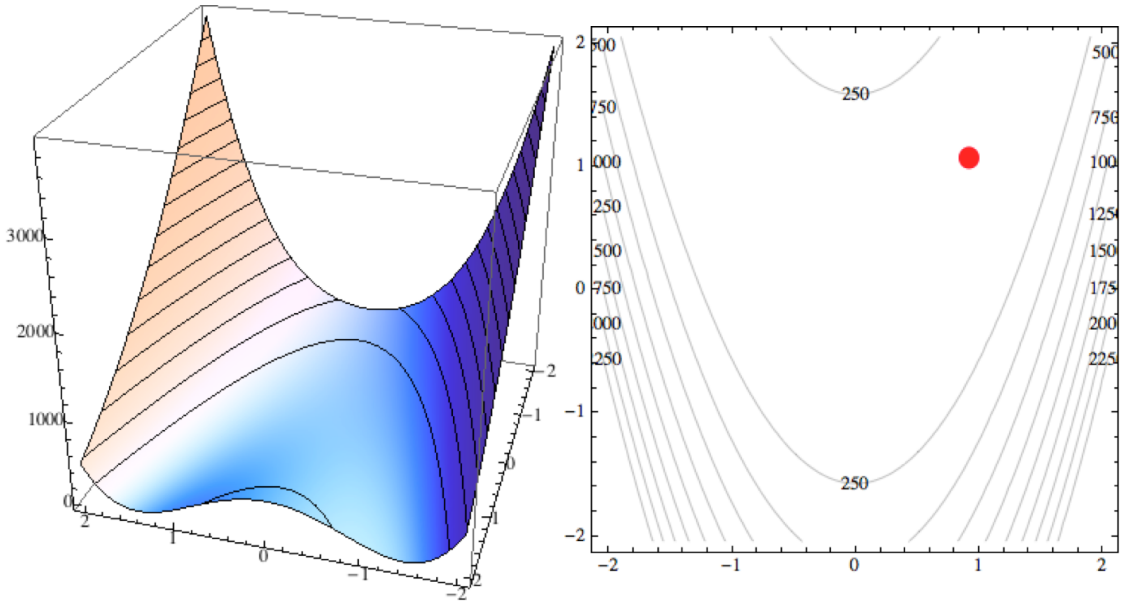
$f(x) = 0$

5.6 Rosenbrockovo sedlo

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1} 100 \left(x_i^4 - x_{i+1}^4\right)^2 + (1 - x_i)^2$$

Průběh funkce:



Obrázek 11: Grafy funkce Rosenbrockovo sedlo
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

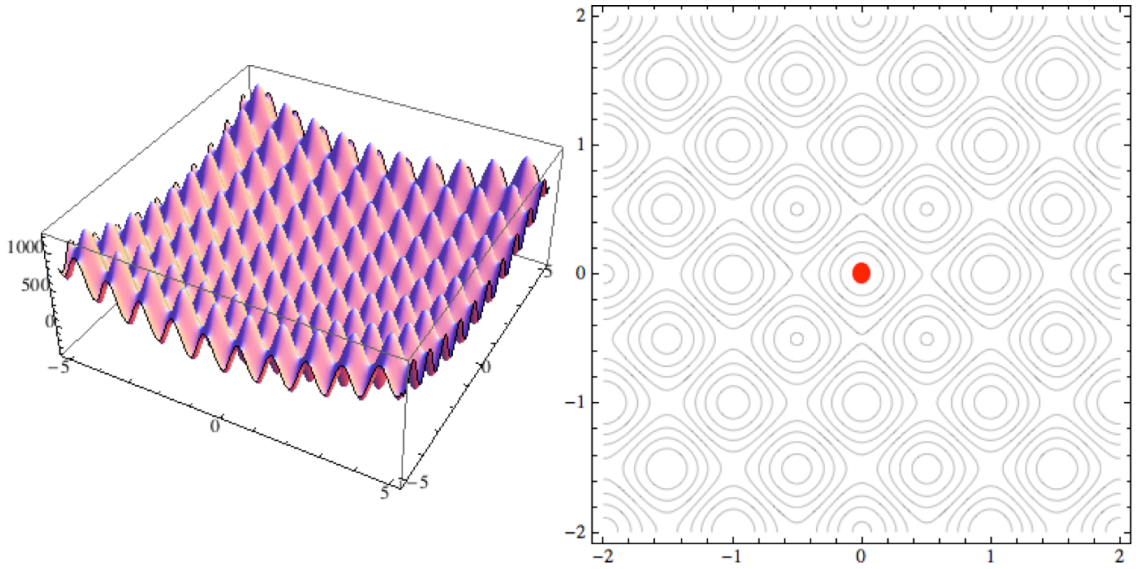
$f(x) = 0$

5.7 Rastriginova funkce

Předpis funkce:

$$f(x) = 2D \sum_{i=1}^{D} x_i^2 - 10\cos(2\pi x_i)$$

Průběh funkce:



Obrázek 12: Grafy Rastriginovy funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = -200 * n$

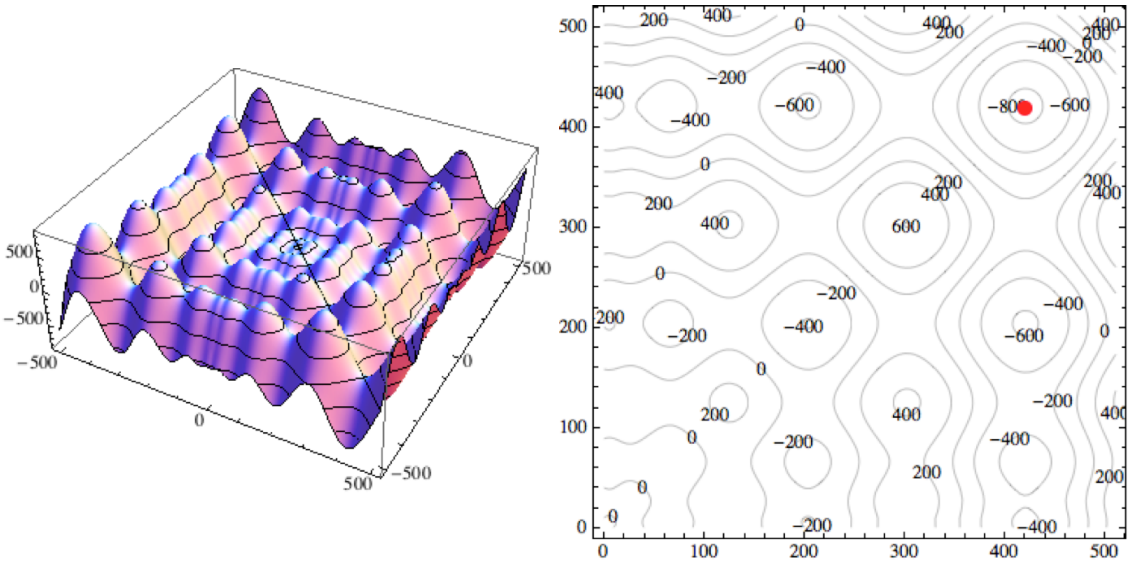
kde n je počet dimenzí prostoru.

5.8 Schwefelova funknce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{|x_i|})$$

Průběh funkce:



Obrázek 13: Grafy Schwefelovy funkce
[zdroj: `http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = -418,983 * n$

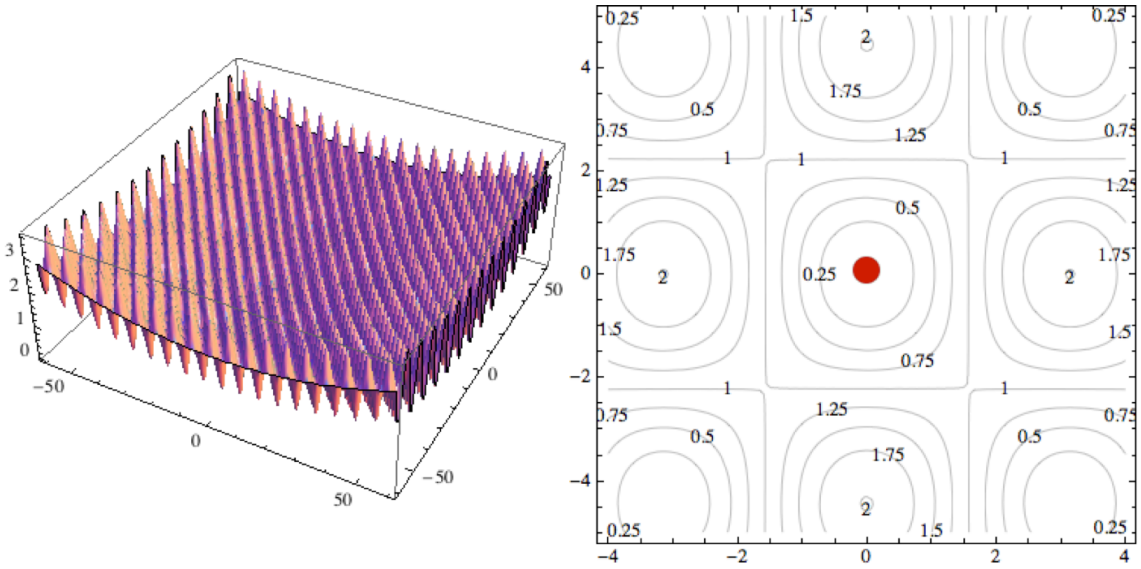
kde n je počet dimenzí prostoru.

5.9 Griewangkova funkce

Předpis funkce:

$$f(x) = 1 + \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}})$$

Průběh funkce:



Obrázek 14: Grafy Griewangovy funkce
[zdroj: `http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

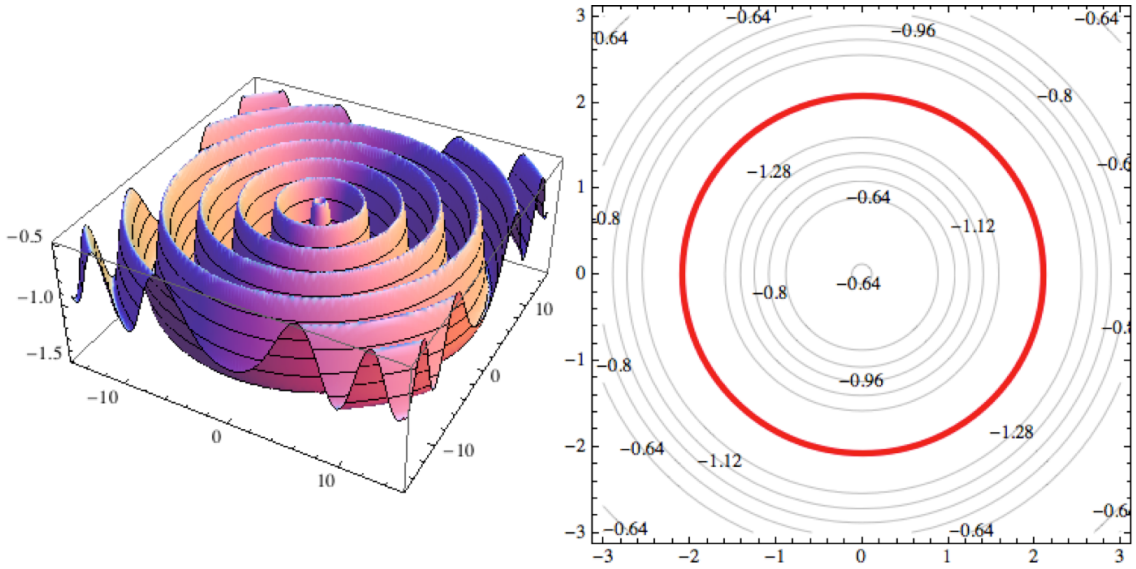
$f(x) = 0$

5.10 Sinová obálková sinusoidální funkce

Předpis funkce:

$$f(x) = -\sum_{i=1}^{D-1} \left(0.5 + \frac{\sin(x_i^2 + x_{i+1}^2 - 0.5)^2}{\left(1 + 0.001\left(x_i^2 + x_{i+1}^2\right)\right)^2} \right)$$

Průběh funkce:



Obrázek 15: Grafy funkce Sinová obálka sinusoidální
[červená kružnice vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = -1,4915 * (n-1)$

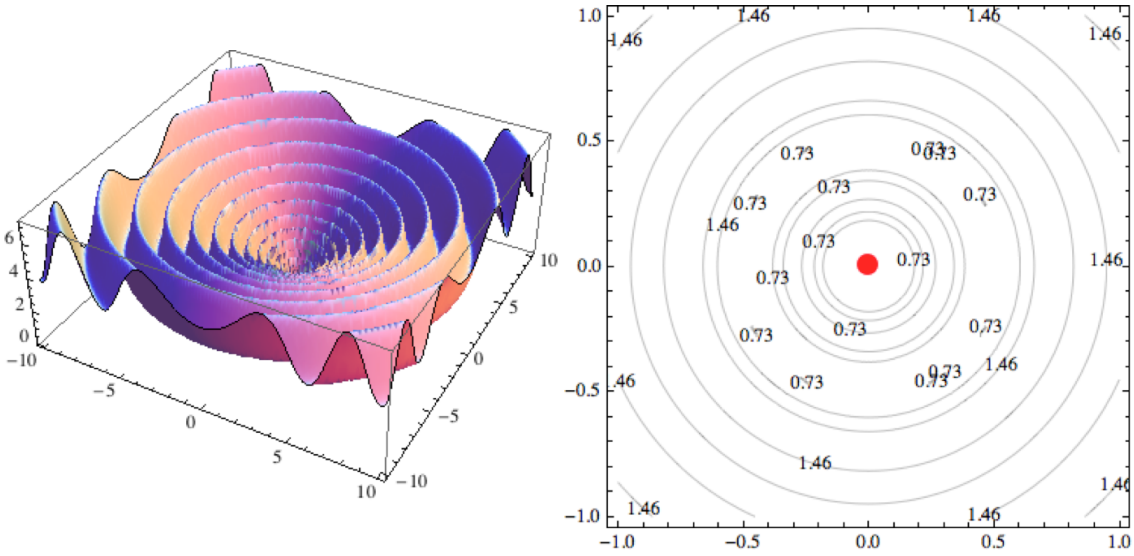
kde n je počet dimenzí prostoru.

5.11 Roztažená sinusoidální V funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1} \left(\sqrt[4]{(x_i^2 + x_{i+1}^2)} \sin(50 \sqrt[10]{x_i^2 + x_{i+1}^2})^2 + 1 \right)$$

Průběh funkce:



Obrázek 16: Grafy Roztažené sinusoidální V funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = 0$

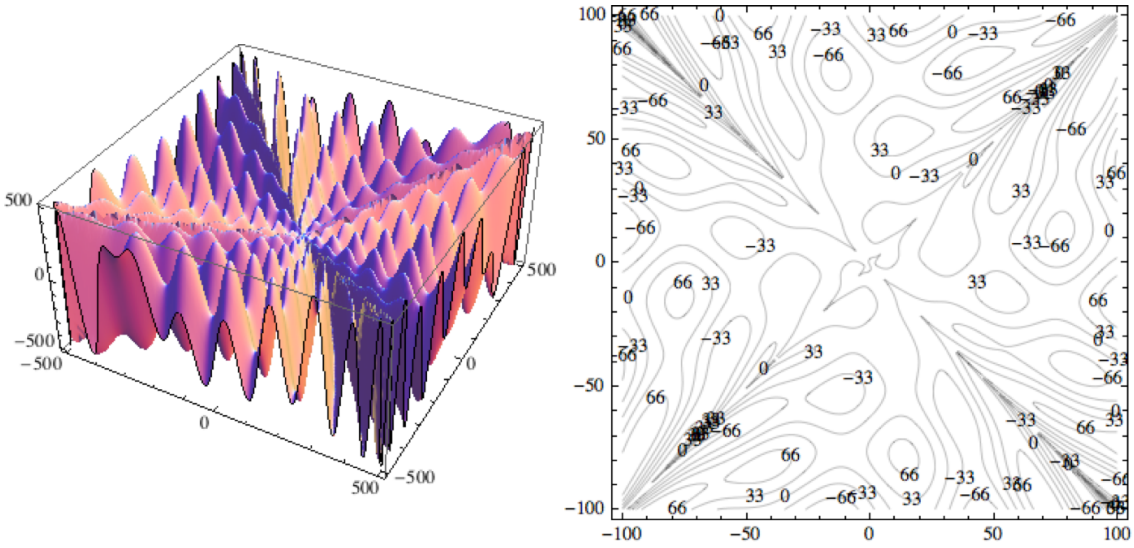
5.12 Ranova funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1}(x_i \sin(\sqrt{|x_{i+1} + 1 - x_i|}) \cos(\sqrt{|x_{i+1} + 1 + x_i|}) +$$

$$(x_{i+1} + 1) \cos(\sqrt{|x_{i+1} + 1 - x_i|}) \sin(\sqrt{|x_{i+1} + 1 + x_i|}))$$

Průběh funkce:



Obrázek 17: Grafy Ranovy funkce

[zdroj: `http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

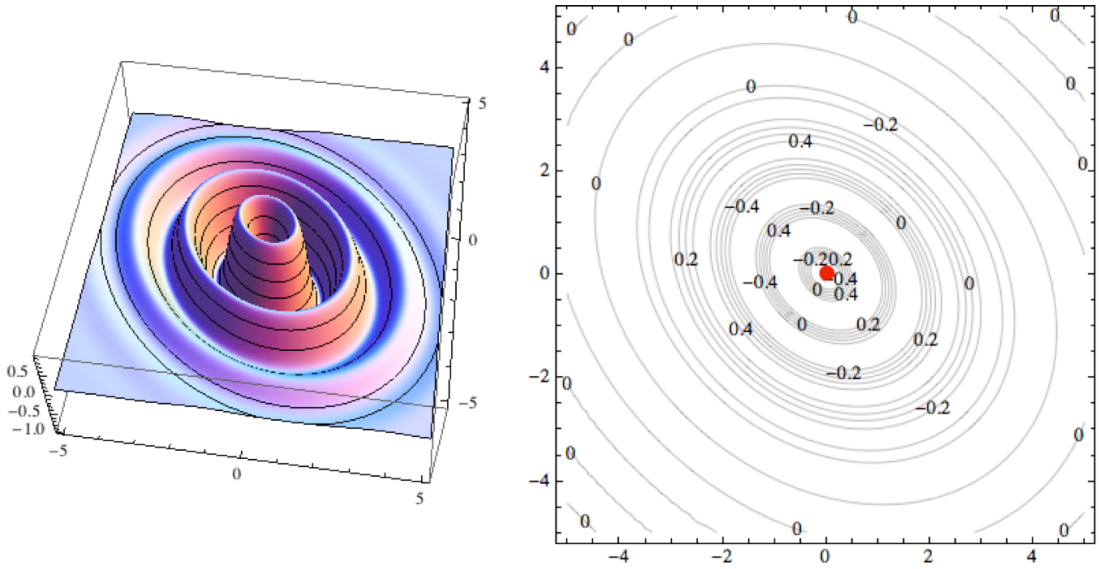
Přesná hodnota globálního minima nebyla v žádné literatuře zjištěna.

5.13 Mastersova funkce

Předpis funkce:

$$f(x) = \sum_{i=1}^{D-1} \left(e^{\frac{\left(x_i^2 + x_{i+1}^2 + 0.5 x_i x_{i+1}\right)}{8}} \cos(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5 x_i x_{i+1}}) \right)$$

Průběh funkce:



Obrázek 18: Grafy Mastersovy funkce
[červený bod vpravo označuje pozici globálního minima, zdroj:
`http://www.ivanzelinka.eu/hp/BIV.html`]

Globální minimum:

$f(x) = -1 * n$

kde n je počet dimenzí prostoru.

6 Výsledky testovaných funkcí

Všechny funkce, popsané v předchozí kapitole, byly předmětem testů SOMA algoritmu. Jejich funkční předpis byl přetransformován do programového kódu, jenž byl umístěn do metody *getFitnessFunction()*.

Pro všechny testy byly nastaveny stejné podmínky, rozumějte parametry algoritmu. Populace byla umístěna do 100D prostoru. Hledaný počet parametrů byl tedy roven 100.

Nalezený globální extrém z jednou proběhnuté optimalizace nebude mít úplnou vypovídající hodnotu. Z toho důvodu bude celý proces spuštěn 10x a výsledné hodnoty nalezených globálních extrémů zprůměrovány.

Ve všech literaturách jsou zpravidla uváděny extrémy typu globální minimum, a tak se zde autor nezabývá nalezením globálního maxima. Ono hledání tohoto opačného extrému spočívá pouze v přidání záporného znaménka před účelovou funkci, respektive jejím vynásobením hodnotou -1.

První sloupec tabulky zmiňuje český název testovací funkce. Druhý poukazuje na procentuální odchylku nalezeného extrému od hodnoty v literaturách. Jsou uváděny jak nejlepší dosažené výsledky, tak i ty nejhorší.

Nastavené hodnoty parametrů pro všechny testovací funkce:

Parametr	**Hodnota**
Mass	3.0
Step	0.11
PRT	0.1
NP	50
D	100
Migrace	200
AcceptedError	0.1

Tabulka 3: Hodnoty parametrů SOMA pro testovací funkce

Testovací funkce	**Odchylka od extrému[%]**	
	Nejlepší	Nejhorší
Ackleyho funkce	0.0	0.5
Plato vajec	-39 502	-34 059
První de Jongova funkce	0.0	0.0001
Třetí de Jongova funkce	0.0	0.0001
Čtvrtá de Jongova funkce	0.0	0.0004
Rosenbrockovo sedlo	0.017	0.031
Rastriginova funkce	0.19	0.55
Schwefelova funkce	0.0	0.23
Griewangkova funkce	0.0001	0.003
Sinova obálková sinusoidální funkce	0	0
Roztažená sinusoidální V funkce	0.34	0.72
Ranova funkce	-23 578	-20 192
Mastersova funkce	1.3	5.8

Tabulka 4: Výsledky testovacích funkcí

Procentuální vzdálenost nalezeného globálního extrému od hodnoty v literaturách se spočte následujícím vztahem:

$$\text{Dist} = 100\frac{|Min_{global} - Min_{nalezene}|}{|Max_{global} - Min_{global}|}[\%]$$

Ve druhém a předposledním řádku jsou uvedeny přímo nalezené hodnoty extrému. Jelikož není v literaturách uvedená naměřená hodnota pro tyto funkce, není s čím porovnávat.

Výsledné hodnoty poukazují na velmi vysokou úspěšnost v přesnosti nalezení globálního minima. I nejhorší výsledek nalezené hodnoty účelové funkce nepřekročil až na jednu výjimku 1 %.

7 SOMA v binárním prostoru

Na rozdíl od např. genetických algoritmů, které pracují s jedinci, jenž jsou reprezentováni binárními hodnotami, byl SOMA navržen tak, aby pracoval v prostoru reálném nebo celočíselném. Tedy pohyb populace se děje v *Euklidovském prostoru*. Pro potřeby komprese, respektive hledání optimální abecedy pro kompresi textových dat, je však potřeba upravit samoorganizující se migrační algoritmus tak, aby byli jedinci reprezentováni binárním vektorem (řetězcem).

Jednou z možností jak tento problém vyřešit, bylo umístit populaci přímo do Hammingova prostoru (binárního prostoru). Tedy prostředí, jež představuje hyperkrychle a vrcholy reprezentují každý z 2^n binárních řetězců délky n. Jedinci by se tak v geometrické interpretaci pohybovali pouze po těchto vrcholech. Bylo však nutné předefinovat význam několika parametrů. Step, neboli krok jedince k vůdci už tak nemohl být realizován přičtením jejich relativní vzdálenosti, jelikož v binárním prostoru nelze dané kauzality využít.

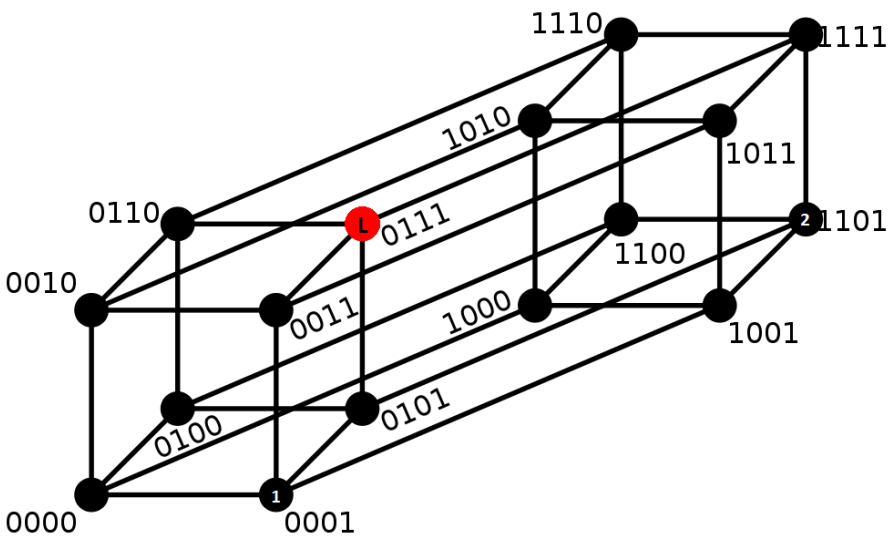
V tabulce 6 a na obrázku 19 je nastíněna zmíněná situace. Jedinec 1 s parametry **P1=0**, **P2=0**, **P3=0**, **P4=1** má možnost k Leaderovi (**0111**) migrovat přes množinu zbylých vrcholů:
$\{0000, 0010, 0011, 0100, 0110, 0101, 1000, 1001, 1010, 1011, 1101, 1110, 1111\}$.
Otázkou však je, kterou cestou má jedinec jít tak, aby vyhovovala geometrickým zákonitostem. Jedinec 1 je od vůdce vzdálen 2 vrcholy, rozumějte, musel by udělat 2 skoky, aby se dostal na jeho pozici. Při obrovské délce binárního řetězce n by však nebylo možné projít každým vrcholem při cestě k vůdci. Situace by ale mohla být změněna tím způsobem, že by uživatel definoval parametr Step tak, aby jedinec dělal skok ob 2, 3, 4,...X_n vrcholů. Obdobně se má situace s parametrem Mass.

Binární řetězec *n=4*	
0000	
0001	Jedinec 1
0010	
0011	
0100	
0101	
0110	
0111	Leader
1000	
1001	
1010	
1011	
1100	
1101	Jedinec 2
1110	
1111	

Tabulka 5: Reprezentace populace v binárním prostoru 1

Pro tento problém bylo využito tzv. Manhattan distance (Taxicab geometry), který říká, že vzdálenost mezi dvěma body je suma absolutních vzdáleností jejich souřadnic[13], na což lze v binárním prostoru nahlížet jako na Hamming distance, neboli Hammingovu vzdálenost. Hammingovu vzdálenost lze definovat jako počet bitů, ve kterých se liší dva binární řetězce stejné délky[14]. Jinak řečeno v geometrické interpretaci, kolik hran je vzdálen jeden vrchol od druhého.

Definice 7.1 *Řekneme, že Hammingova vzdálenost je vzdálenost mezi binárními řetězci A a B, která se spočte* $\sum_{i=1}^{n} |A_i - B_i|$



Obrázek 19: Možná geometrická interpretace 4D binárního prostoru
[Červenou barvou označen Leader. Čísly 1 a 2 zbylí jedinci, zdroj:
`http://http://en.wikipedia.org/wiki/Hamming_distance`]

Příklad 7.1
Mějme binární řetězce A = 0100100 a B = 1101101. Hammingova vzdálenost mezi nimi je: $HD = |0-1| + |1-1| + |0-0| + |0-1| + |1-1| + |0-0| + |0-1|$

$HD = 3$

A = **0**10**0**10**0**
B = **1**10**1**10**1**

Řetězec A se od řetězce B liší ve 3 bitech. ■

Jestliže budou bity, v nichž se řetězce liší podle určitých pravidel, měněny, dostaneme se v pár krocích z jedné pozice do druhé. Parametr Step pak bude vyjadřovat procento změny bitů a bude tak předefinován jeho rozsah na $\langle 0.0, 1.0 \rangle$.

Parametr Mass zde potom vyjadřuje, v jaké vzdálenosti se zastaví jedinec od vůdce ve smyslu počtu hran.

Takto naimplementovaný algoritmus však nevykazoval příliš dobré výkonostní vlastnosti, co se nalezení globálního extrému týče. Byla provedena řada testů na nichž se ukázalo, že tímto způsobem není ve většině případů globální extrém nalezen příliš přesně. Bylo tak potřeba nalézt jinou variantu, která by zajistila kvalitnější řešení.

Algoritmus SOMA má nejlepší výsledky, když pracuje v prostoru reálných čísel. Pokud je potřeba jiné číselné soustavy, je nejlepší dostat se k takovým hodnotám převodem nebo jinou funkcí. V tomto případě je využito zaokrouhlení na celá čísla.

Algoritmus je ponechán v běhu reálných hodnot s tím, že hranice specimenu jsou nastaveny od 0 po 1 pro všechny použité parametry. Celý proces funguje obdobně, jak bylo popsáno v kapitole 3. Při výpočtu hodnoty účelové funkce je však zavolána matematická funkce *Math.Round(reálná hodnota)* v C#, jež nám vrátí zaokrouhlenou hodnotu jednoho parametru. Tento postup je opakován na všechny zbývající parametry a hodnoty jsou poté poskládány do binárního řetězce, respektive vektoru binárních hodnot. Takto zpracovaný se dále využije pro hledání abecedy pro kompresi textu.

	P1	P2	P3	P4	P5	P6
Jedinec 1	0.111	0.454	0.891	0.322	0.650	0.999
Zaokrouhlení	0.0	0.0	1.0	0.0	1.0	1.0
Výsledný binární vektor ($n = 6$)	0	0	1	0	1	1

Tabulka 6: Reprezentace populace v binárním prostoru 2

Testování proběhlo také v případě této varianty a výsledky jasně poukazují na velkou úspěšnost v přesném nalezení globálního extrému. Také z časového hlediska v porovnání s prvním případem vychází lépe tato realizace. Při malých délkách binárních řetězců jsou časy srovnatelné, avšak pro $n > 400$ jsou hodnoty měření řádově horší.

Poznámka 7.1 Výsledky testů obou verzí implementace nejsou uváděny z důvodu irelevance pro tuto práci.

8 Komprese dat

I v dnešní době pevných velkokapacitních, přenosných flash, nebo již upadajících optických médií je otázka stlačení dat na menší velikost velmi aktuální, ačkoliv zařízení pro ukládání dat doznaly za posledních 10 let obrovského nárůstu kapacity. Lidé nesnáší čekání na to, než se jim načte webová stránka, nebo než se stáhne soubor ze serveru. Každá taková sekunda čekání navíc připadne jako celé století.

Proces transformace zdroje dat do menší velikosti se nazývá komprese. Hlavní motivací pro tento obor je **ukládání dat** na média s omezenou kapacitou, snížení nároků na objem **přenesených dat** a také omezení prostoru v **archivu**, který zabírají málo používaná data.

Zpětná obnova původní podoby je dekomprese. Typickým zdrojem dat je řetězec nebo posloupnost bajtů. Teorie komprese dat je starší než éra počítačů.

8.1 Princip

Komprese je založena na odstraňování redundance. Jinak řečeno zdrojová data obsahují nadbytečné informace, které lze vyjádřit v jiném (úspornějším) formátu. Nadbytečné informace spočívají především v :

- opakování symbolů
- opakování slov či frází z těchto symbolů složených
- kontextovou závislostí (např. v anglických slovech se za písmenem *Q* většinou objevuje *U*)
- neefektivní přímou reprezentací
- nejednotným rozložením symbolů v textu

Při komprimaci se snažíme dosáhnout co nejlepšího poměru mezi délkou zprávy a její informační hodnotou

8.2 Druhy komprese

- **Ztrátová** - využívá nedokonalosti lidských smyslů. Je pouhou aproximací originálu - z informace odstraňuje detaily do takové míry, dokud ji lze uspokojivě rekonstruovat. Tento způsob komprese se nejčastěji používá pro multimediální formáty dat. To znamená pro hudební skladby, video a obrázky. Příkladem známých formátů využívajících lidských nedokonalostí je např. JPEG, H.264, MPEG, MP3.
- **Bezeztrátová** - zachovává vždy kompletní informaci. Jinými slovy dekomprimovaná data jsou vždy shodná s originálem. Nejčastější použití je pro soubory nesoucí nějaký typ psané informace, jako textové dokumenty, zdrojové kódy apod. Jen těžko by bylo možné odstranit například z textové zprávy některá slova. Je totiž nemožné vybrat slova, která nejsou pro význam informace důležitá. Typickými zástupci této kategorie jsou algoritmy LZ77, LZ78, LZW, Huffmanovo kódování. Mezi formáty patří ZIP, RAR, gzip.

8.3 Základní metody bezeztrátové komprese

Rozdělení metod pro bezeztrátovou kompresi může být následující:

- **Statistické metody** - jsou založené na pravděpodobnosti výskytu jednotlivých symbolů. *(Huffmanovo Kódování, Shannon-Fanovo kódování)*
- **Slovníkové metody** - používají slovník jako dynamickou strukturu, která se postupně tvoří během čtení vstupu. *(LZ77, LZ78, LZW, LZMA)*
- **Transformace** - algoritmy spadající do této kategorie neprovádějí kompresi, avšak modifikují data tak, aby se dala lépe zkomprimovat. *(Burrows-Wheelerova transformace, Move-to-front transformace)*

8.4 Kompresní poměr

Efektivita komprese je různá a záleží na typu dat. Text a grafika mohou být obvykle zkomprimovány velkou měrou.

Běžný způsob měření účinnosti komprese je označován jako kompresní poměr. Je to poměr velikostí originálního souboru se souborem po komprimaci a je udáván v procentech.

$$\mathbf{KP} = \frac{\text{velikost komprimovaného souboru}}{\text{velikost originálního souboru}} * 100[\%]$$

Takto definovaný vztah popisuje, kolik procent z původní velikosti zabírá soubor po kompresi.

9 Realizace hledání optimální abecedy

V této části je popsán praktický přístup autora k nalezení optimální abecedy.

9.1 Typy přístupů k textovým datům

U většiny běžných komprimačních algoritmů dochází ke čtení vstupního souboru bajt po bajtu, jinak řečeno **znak** po znaku a poté se zpracovává určitou metodou. U souborů s velkým objemem čistého textu (cca 1 000 - 100 000 slov) už však není tento přístup docela výhodný. V takovýchto případech je efektivnější k souborům přistupovat po **slabikách**, nebo také po celých **slovech**. Tyto 3 různé techniky jsou však vždy použity odděleně. Jinými slovy nikdy se jeden přístup neprolíná s druhým. Mohlo by být však velice efektivní využít například přístup po znacích a zároveň mít také možnost využít pár slov, které se v textu často opakují.

9.1.1 Přístup po znacích

Soubor je „rozparsován“ na jednotlivé znaky, které se vkládají do slovníku za podmínky, že už v něm není obsažen. Implementace algoritmu spočívá v procházení celého souboru znak po znaku a dělání si záznamů o existujících znacích v datové struktuře představující slovník.

9.1.2 Přístup po slabikách

Na světě se hovoří převážně 11 jazyky. Každý z těchto jazyků má svoje různá specifika. Morfologie slov je naprosto odlišná. V českém jazyce může mít jedno slovo až 20 různých tvarů, a slabikové rozložení bude pro každý takový tvar jiné. Je tedy nemožné zobecnit metody pro rozdělení na slabiky. Hlavním důvodem je však především nejednoznačnost rozdělení jednotlivých slov na slabiky. Kdybychom si vzali například slovo Ostrava. Z pohledu českého jazyka jsou možná dvě rozdělení, a to: **Ost-ra-va** nebo **Os-tra-va**. Dalším aspektem je sémantika slov. Mohou existovat slova začínající stejným sledem písmen, avšak ve významu se lišící. Slova **neu-ron** a **ne-u-ro-nit** jsou důkazem jiného rozdělení slabik pro různé významy. Tento přístup tedy není v praxi hojně používaný, a ani v této práci se autor implementací algoritmu řešící tuto problematiku nezabývá.

9.1.3 Přístup po slovech

U tohoto přístupu je potřeba vybrat slova bez duplikátů. Slovník poté obsahuje pouze tvary bez redundancí. Při implementaci v programovacím jazyce je třeba si definovat znaky, jenž budou považovány za slovní oddělovače. Poté je soubor opět sekvenčně procházen a do datové struktury slovníku jsou vkládána nově nalezená slova. V jazyce C# existuje varianta s funkcí *Split*. Na ukázce zdrojového kódu jsou jako pole znaků definované oddělovače a poté je na *text* zavolána funkce Split, která naplní pole řetězců nalezenými slovy.

```
char[] delimiterChars = { ' ', ',', '.', ':', '\t', '\r', '\n' };
string[] words = text.Split(delimiterChars);
```

Výpis 5: Dekompozice textu na slova

9.1.4 Přístup po x-gramech

Výrazem x-gram se myslí sekvence znaků délky **x**. Tímto způsobem se tak v jistém smyslu nahrazuje výše popsaný přístup po slabikách a teoreticky by se tímto dala nahradit také slova. Pro příklad, kdybychom chtěli vypsat všechny 2-gramy ze vstupního textu „ahoj“ vypadala by množina následovně:

$\{ah, ho, oj\}$ s překryvem písmen.
$\{ah, oj\}$ bez překryvu.

9.2 Proces hledání optimální abecedy

V této práci se autor zabývá problémem skloubení výše zmíněných variant přístupu tak, aby měly pozitivní dopad na kompresní poměr. Jelikož se jedná o velmi komplikovaný problém, není možnost všechny varianty vyzkoušet sekvenčně. Pokud bychom totiž měli ve slovníku všechna písmena, slabiky (popř. x-gramy) a slova, dostali bychom se na 2^n možností, kde n je počet symbolů ve slovníku, což by bylo obrovsky časově náročné.

Matematická optimalizace v tomto případě také nepřichází v úvahu, jelikož namodelování rovnice, řešící tuto problematiku je takřka nemožné. Na řadu tedy

přichází bio-optimalizace, konkrétně bude tedy použit výše popsaný a otestovaný SOMA algoritmus.

Po úpravě do binárního řešení, respektive takové úpravě, aby byl jedinec reprezentován binárním řetězcem, viz kapitola 7, je algoritmus připraven pro potřeby problému hledání optimální abecedy.

Jedinec bude stejné délky jako počet položek ve slovníku. Tím se namapuje každý bit binárního řetězce na určitý symbol. Když se poté na i-té pozici v binárním řetězci objeví 1 (true), znamená to, že se i-tý symbol ve slovníku použije. V případě hodnoty 0 (false) se provede opak.

Slovník	ab	cf	poe	r	er	bvm	a	e
Jedinec	0	1	0	1	1	0	1	0

Tabulka 7: Selekce symbolů

Pro aktuální iteraci by byly dostupné symboly $\{cf, r, er, a\}$

9.3 Realizace slovníku

Slovník je realizován generickým datovým typem $List\langle T\rangle$, který je plněn objekty, jenž obsahují informace o **symbolu** a jeho **délce**.

9.4 Transformace vstupního textu

Když je inicializován slovník s povolenými symboly, přichází na řadu převyprávění souboru pomocí celočíselných hodnot. Postupně se prochází vstupní text a pokud je načtená sekvence znaků nalezena ve slovníku, tak je nahrazena celým číslem, značícím jeho pozici. Po ukončení celého procesu dostaneme výstup ve formě sekvence celých čísel.

Příklad 9.1
Slovník :

Slovník	t	,či	␣ne	r	bý	bvm	a	e
Pozice	0	1	2	3	4	5	6	7

Tabulka 8: Příklad slovníku

Vstup: být,či nebýt
Výstup: 4 0 1 2 4 0 ■

9.5 Definice účelové funkce

Jelikož je komprese provedena na optimální abecedou transformovaném textu, je potřeba k velikosti komprimovaného souboru připočíst také velikost nalezené abecedy. Tato abeceda bude totiž muset být vždy uložena spolu s komprimovaným řetězcem pro zpětnou transformaci do originální podoby textu. Fitness funkce je proto definována jako velikost komprimovaného souboru + velikost optimalizované abecedy. Úkolem vybraného optimalizačního algoritmu je nalezení nejmenší takové hodnoty.

9.6 Kompresní algoritmus

Jako algoritmus obstarávající kompresi byl vybrán LZW. Hlavním důvodem této volby je rychlost procesu komprimace a také jednoduchost implementace. Algoritmu jsou předávány určité informace ohledně použité abecedy.

10 Experimenty

V této předposlední kapitole se autor zabývá testováním vytvořeného algoritmu nad různým typem textových souborů.

10.1 Testovací data

Testovací data jsou pouze textového typu. Jsou vybrány soubory z množiny připravených pro potřeby komprese a jejich globální porovnávání. První sekce se nazývá *Artificial corpus*. Jsou to soubory, které nemají příliš velkou informační hodnotu. Druhá sekce, obsahující soubory různého typu, má název *Canterbury corpus*. Poslední *Large corpus*, poskytuje soubory velkého objemu dat.

Artificial corpus

- **aaa.txt** - písmeno „a“ opakované 100 000 krát.
- **alphabet.txt** - všechna písmena anglické abecedy opakovaná do délky 100 000.

Canterbury corpus

- **grammar.lsp** - zdrojový kód napsaný v programovacím jazyce LISP.
- **xargs.1** - manuál k operačnímu systému GNU.
- **fields.c** - zdrojový kód programu v jazyce C.
- **cp.html** - HTML zdrojový kód.
- **asyoulik.txt** - komediální dílo Williama Shakespeara „As You Like It“.
- **alice29.txt** - novela Ch.L. Dodgsona o Alence, která padala králičí norou.
- **Icet10.txt** - technický dokument.
- **plrabn12.txt** - básnické dílo Johna Miltona.

Large corpus

- **world192.txt** - souhrn statistických geografických, demografických a jiných dat, spravovaný americkou agenturou CIA.
- **bible.txt** - Bible, verze Kinga Jamese.

10.2 Analýza dat

U všech vybraných souborů ze zmíněných tří sekcí je provedena analýza, pro zjištění počtu symbolů v nich. Pro testování byly vybrány znaky(1-gramy), 2-gramy, 3-gramy, 4- gramy, 5-gramy a slova. Následující tabulka obsahuje hojnost výskytu ve všech kategoriích bez opakování. U všech jsou vybírány pouze symboly bez překryvu.

V prvním sloupci tabulky se nachází velikost originálních dat. V následujících šesti sloupcích je zmíněna četnost symbolů pro dané soubory a poslední obsahuje jejich celkový součet.

Soubor	Velikost [B]	1-gramy	2-gramy	3-gramy	4-gramy	5-gramy	Slova	Suma
aaa.txt	100 000	1	1	1	1	1	1	5
alphabet.txt	100 000	26	13	26	13	26	1	105
grammar.lsp	3 721	76	354	526	539	523	247	2 265
xargs.1	4 227	74	442	743	761	683	303	3006
fields.c	11 150	90	645	1 135	1 347	1 376	435	5 028
cp.html	24 603	86	1 192	2 803	3 019	2 843	1 206	11 149
asyoulik.txt	125 179	68	1 043	4 730	10 269	14 050	4 201	34 361
alice29.txt	152 089	74	1 132	4 759	10 109	14 277	4 004	34 355
Icet10.txt	426 754	84	1 714	7 672	18 044	28 675	7 644	63 833
plrabn12.txt	481 861	81	1 088	6 499	19 203	34 999	12 966	74 836
world192.txt	2 473 400	94	2 779	20 615	60 684	99 986	30 663	214 821
bible.txt	4 047 392	82	1 585	10 109	36 089	83 851	20 687	152 403

Tabulka 9: Analýza testovacích dat

10.3 Výsledky testů

Pro testování je vybráno 8 různých kombinací symbolů ze zmíněných šesti kategorií pro zlepšení komprese. Jsou to spojení: *1-gramy a 2-gramy*, *1-gramy a 3-gramy*, *1-gramy a 4-gramy*, *1-gramy a 5-gramy*, *1-gramy a slova*, *1-gramy, 2-gramy a 3-gramy*, *1-gramy, 2-gramy a 5-gramy*, *1-gramy, 2-gramy a slova* a samostatně je ještě pro porovnání otestována 1-gramová komprese. U jednotlivých testů není počítáno se všemi existujícími symboly, ale je vybrána pouze jejich část. U menších souborů se jedná o jednu třetinu nejčetnějších z dané kategorie a u větších se vybere přesné množství 250-500 nejfrekventovanějších.

U souborů do velikosti 100 KB byly provedeny všechny testy 5x, u souborů do 1 MB 2x a výsledné hodnoty jsou zprůměrovány. Soubory bible.txt a world192.txt pouze jednou z důvodu výpočetní náročnosti.

V tabulkách pro jednotlivé soubory jsou zaznačeny výsledky testování. První sloupec je vyhrazen pro typ kombinace symbolů. Druhý sloupec obsahuje průměrnou hodnotu nejmenší nalezené velikosti souborů po kompresi. Ve třetím sloupci se nachází kompresní poměr a ve čtvrtém průměrný počet symbolů potřebný pro transformaci dat. Poslední sloupec poukazuje na rozdíl mezi běžnou, 1-gramovou kompresí a kompresí pomocí abecedy nalezené evolučním algoritmem.

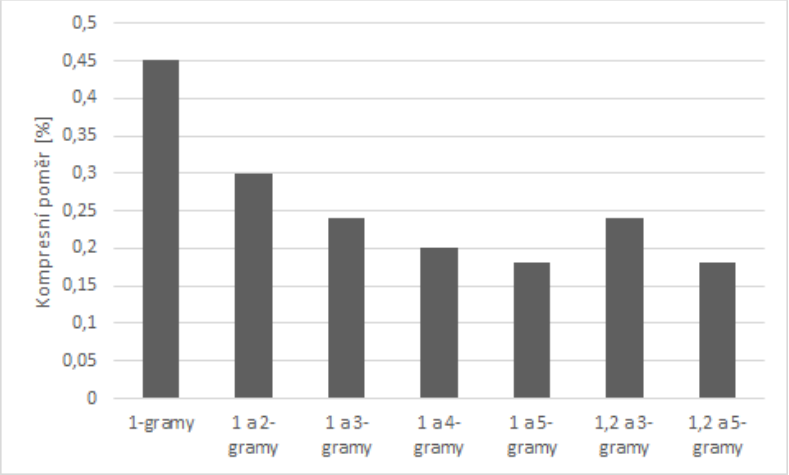
V grafech jsou namodelovány jednotlivé kompresní poměry v závislosti na druhu kombinace symbolů.

V poslední části jsou zmíněny konfigurace samoseorganizujícího migračního algoritmu použité pro testovací případy.

10.3.1 aaa.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	100 000	x	x	x
1-gramy	451	0.45	1	x
1 a 2-gramy	304	0.30	1	0.15
1 a 3-gramy	244	0.24	2	0.21
1 a 4-gramy	206	0.20	1	0.25
1 a 5-gramy	183	0.18	1	0.27
1-gramy a slova	x	x	x	x
1,2 a 3-gramy	244	0.24	2	0.21
1,2 a 5-gramy	183	0.18	1	27
1,2-gramy a slova	x	x	x	x

Tabulka 10: Výsledky testování - aaa.txt



Obrázek 20: Dosažené kompresní poměry - aaa.txt

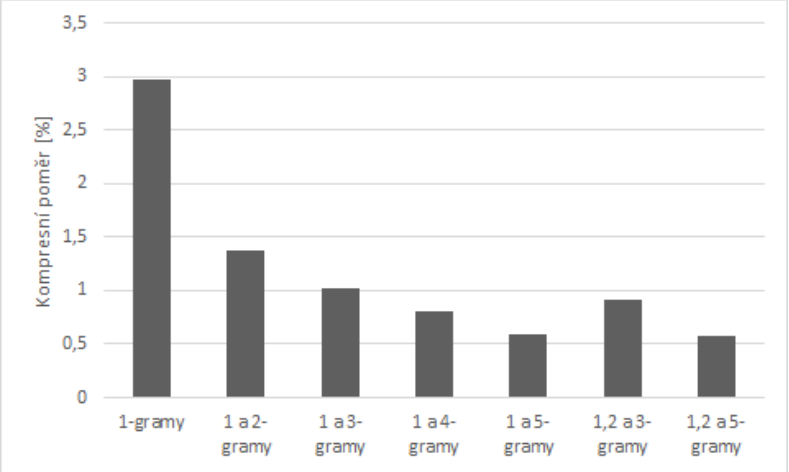
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.33, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.2 alphabet.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	100 000	x	x	x
1-gramy	2 980	2.98	26	x
1 a 2-gramy	1 366	1.37	13	1.61
1 a 3-gramy	1017	1.02	10	1.96
1 a 4-gramy	795	0.80	8	2.18
1 a 5-gramy	586	0.59	9	2.39
1-gramy a slova	x	x	x	x
1,2 a 3-gramy	910	0.91	10	2.07
1,2 a 5-gramy	582	0.58	7	2.4
1,2-gramy a slova	x	x	x	x

Tabulka 11: Výsledky testování - alphabet.txt



Obrázek 21: Dosažené kompresní poměry - alphabet.txt

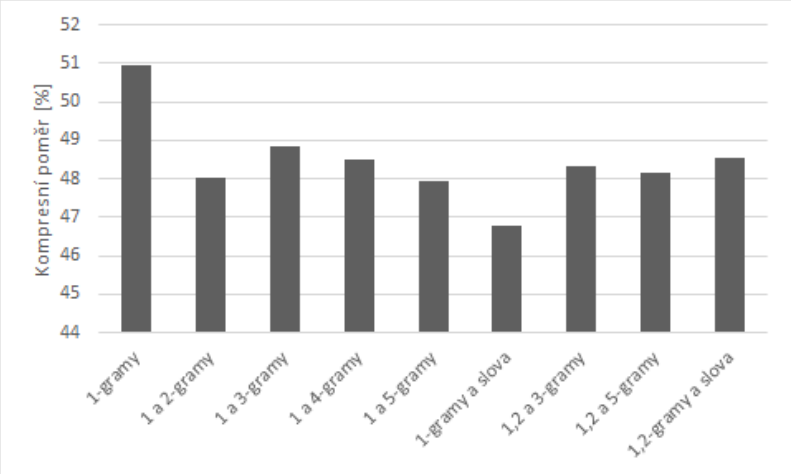
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.33, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.3 grammar.lsp

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	3 721	x	x	x
1-gramy	1895	50.93	76	x
1 a 2-gramy	1786.8	48.02	102.6	2.91
1 a 3-gramy	1817	48.83	112	2.1
1 a 4-gramy	1805.2	48.51	100.4	2.42
1 a 5-gramy	1783	47.92	97	3.01
1-gramy a slova	1741.4	46.80	102.6	4.13
1,2 a 3-gramy	1797.6	48.31	123.8	2.62
1,2 a 5-gramy	1791.6	48.15	111.4	2.78
1,2-gramy a slova	1807	48.56	127.4	2.37

Tabulka 12: Výsledky testování - grammar.lsp



Obrázek 22: Dosažené kompresní poměry - grammar.lsp

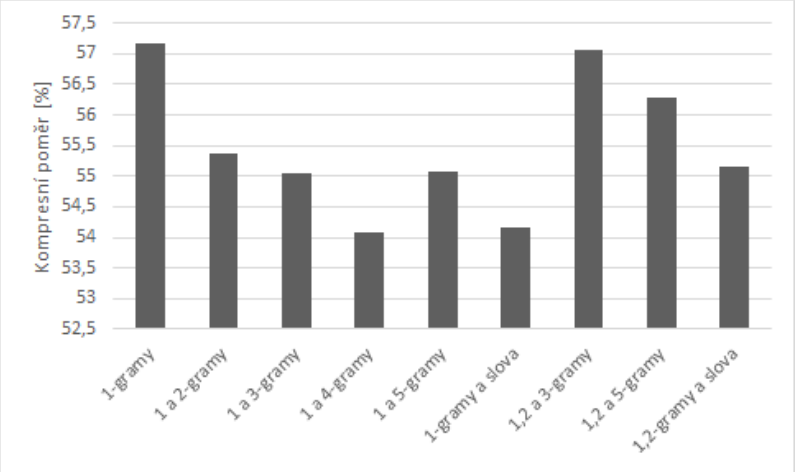
Všechny testy byly provedeny s konfigurací SOMA parametrů :

```
Mass: 3.0, Step: 0.22, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.4 xargs.1

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	4 227	x	x	x
1-gramy	2416	57.16	74	x
1 a 2-gramy	2341	55.38	122.6	1.78
1 a 3-gramy	2326.8	55.05	111.2	2.11
1 a 4-gramy	2286	54.08	104	3.08
1 a 5-gramy	2327.8	55.07	105.6	2.09
1-gramy a slova	2289.2	54.16	104	3.0
1,2 a 3-gramy	2412	57.06	167	0.10
1,2 a 5-gramy	2379	56.28	144.2	0.88
1,2-gramy a slova	2331.2	55.15	134.8	2.01

Tabulka 13: Výsledky testování - xargs.1



Obrázek 23: Dosažené kompresní poměry - xargs.1

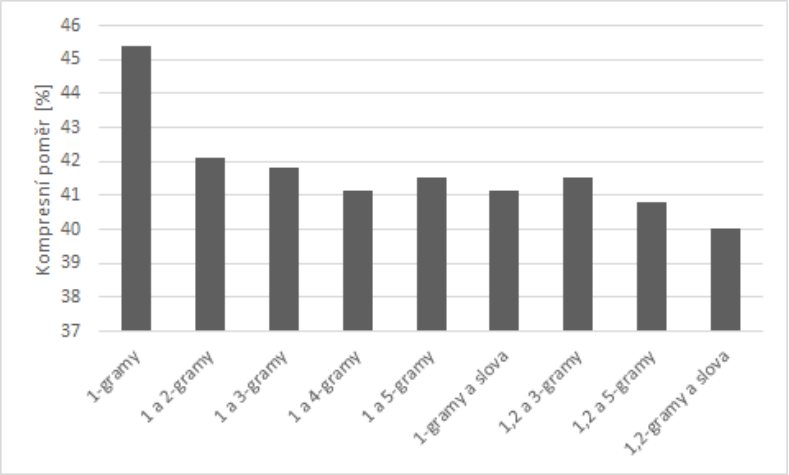
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.11, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.5 fields.c

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	11 150	x	x	x
1-gramy	5 062	45.40	90	x
1 a 2-gramy	4693.8	42.10	165.2	3.3
1 a 3-gramy	4665.2	41.84	132	3.56
1 a 4-gramy	4587	41.14	122.4	4.26
1 a 5-gramy	4628.2	41.51	117	3.89
1-gramy a slova	4585	41.12	139.4	4.28
1,2 a 3-gramy	4630	41.52	159	3.88
1,2 a 5-gramy	4544.8	40.80	150	4.6
1,2-gramy a slova	4463	40.03	170	5.37

Tabulka 14: Výsledky testování - fields.c



Obrázek 24: Dosažené kompresní poměry - fields.c

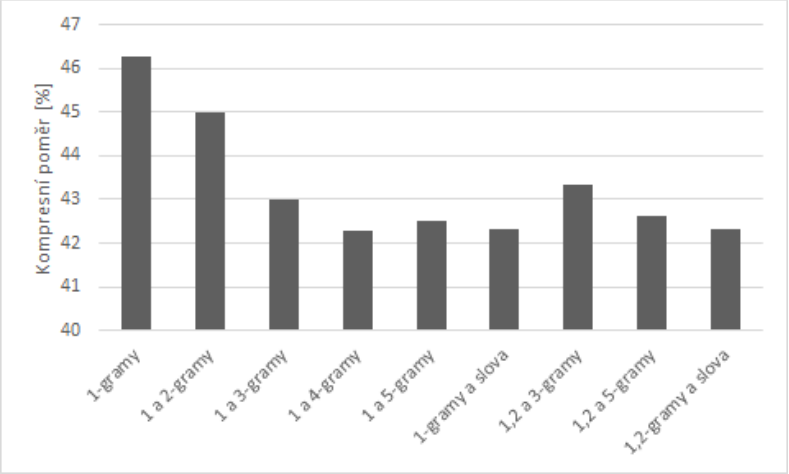
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.11, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.6 cp.html

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	24 603	x	x	x
1-gramy	11 384	46.27	86	x
1 a 2-gramy	11 072	45.00	238	1.27
1 a 3-gramy	10 578.2	43.00	134.6	3.27
1 a 4-gramy	10 408	42.30	131	3.97
1 a 5-gramy	10 456.4	42.50	126	3.77
1-gramy a slova	10 410	42.31	177.2	3.96
1,2 a 3-gramy	10 663	43.34	191.2	2.93
1,2 a 5-gramy	10 484.8	42.62	166.2	3.65
1,2-gramy a slova	10 410	42.31	138.2	3.96

Tabulka 15: Výsledky testování - cp.html



Obrázek 25: Dosažené kompresní poměry - cp.html

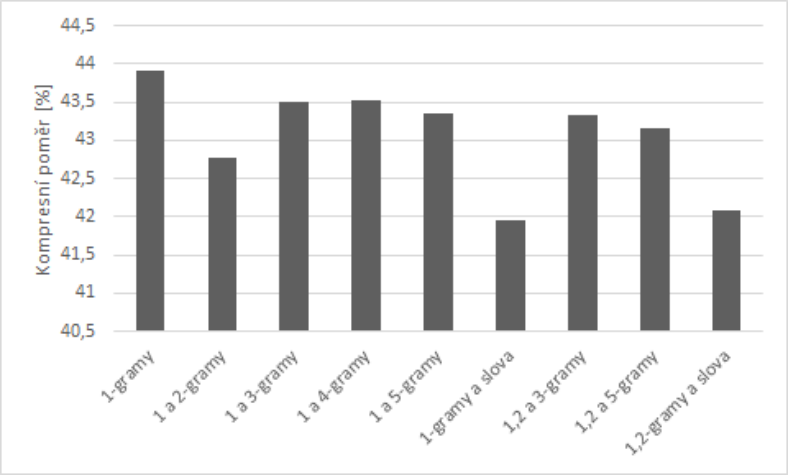
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.11, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.7 asyoulik.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	125 179	x	x	x
1-gramy	54 964	43.91	68	x
1 a 2-gramy	53 531	42.76	202	1.15
1 a 3-gramy	54 466.5	43.51	207.5	0.4
1 a 4-gramy	54 487.5	43.53	273	0.38
1 a 5-gramy	54 262.5	43.35	279.5	0.56
1-gramy a slova	52 524	41.96	213	1.95
1,2 a 3-gramy	54 234.5	43.33	280	0.58
1,2 a 5-gramy	54 022	43.16	268.5	0.75
1,2-gramy a slova	52 683.5	42.09	277.5	1.82

Tabulka 16: Výsledky testování - asyoulik.txt



Obrázek 26: Dosažené kompresní poměry - asyoulik.txt

Testy byly provedeny s dvěma konfiguracemi SOMA parametrů:

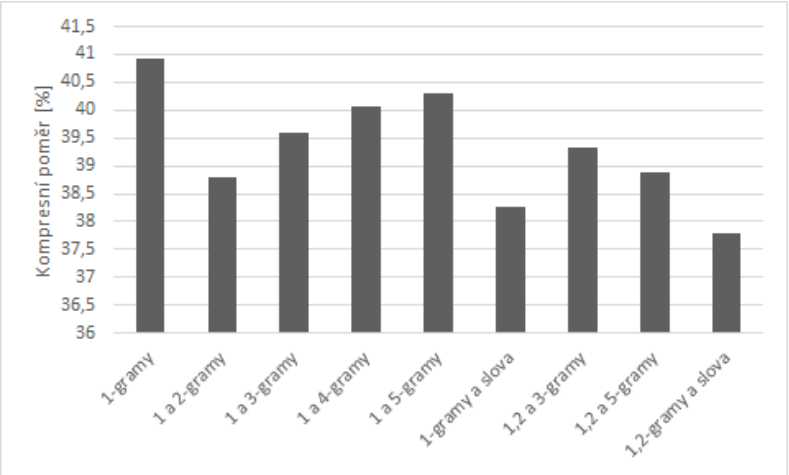
```
Mass: 3.0, Step: 0.33, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

```
Mass: 3.0, Step: 0.66, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.8 alice29.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	152 089	x	x	x
1-gramy	62 219	40.91	74	x
1 a 2-gramy	59 003.5	38.80	246	2.11
1 a 3-gramy	60 216.5	39.59	189.5	1.32
1 a 4-gramy	60 955.5	40.08	309	0.83
1 a 5-gramy	61 276	40.29	300.5	0.62
1-gramy a slova	58 197	38.27	230.5	2.64
1,2 a 3-gramy	59 832.5	39.34	294.5	1.57
1,2 a 5-gramy	59 164	38.90	301.5	2.01
1,2-gramy a slova	57 494	37.80	313	3.11

Tabulka 17: Výsledky testování - alice29.txt



Obrázek 27: Dosažené kompresní poměry - alice29.txt

Testy byly provedeny s dvěma konfiguracemi SOMA parametrů:

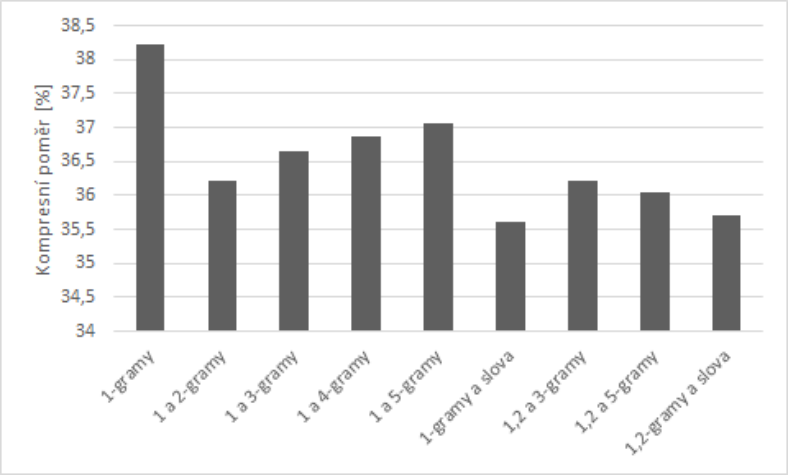
```
Mass: 3.0, Step: 0.33, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

```
Mass: 3.0, Step: 0.66, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.9 Icet10.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	426 754	x	x	x
1-gramy	163 156	38.23	84	x
1 a 2-gramy	154 576	36.22	245.5	2.01
1 a 3-gramy	156 366	36.64	195	1.59
1 a 4-gramy	157 289	36.86	333	1.37
1 a 5-gramy	158 202	37.07	311.5	1.16
1-gramy a slova	151 995	35.62	222.5	2.61
1,2 a 3-gramy	154 555	36.22	318.5	2.01
1,2 a 5-gramy	153 798	36.04	297.5	2.19
1,2-gramy a slova	152 333.5	35.70	266	2.53

Tabulka 18: Výsledky testování - Icet10.txt



Obrázek 28: Dosažené kompresní poměry - Icet10.txt

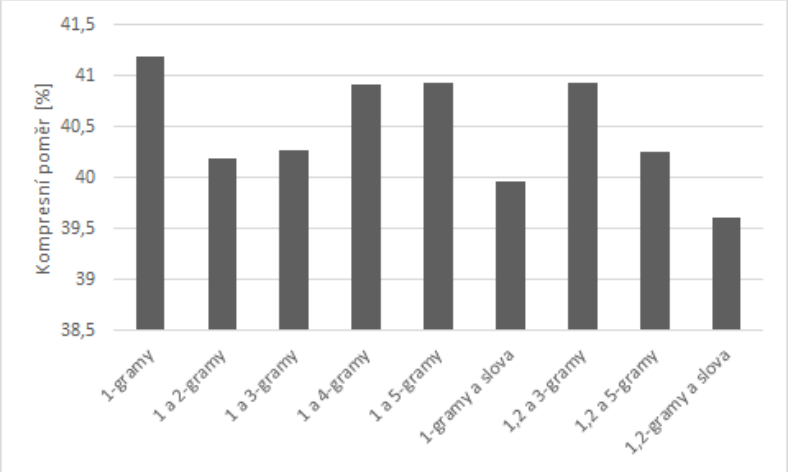
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.66, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.10 plrabn12.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	481 861	x	x	x
1-gramy	198 441	41.18	81	x
1 a 2-gramy	193 590	40.18	188.5	1.0
1 a 3-gramy	194 064	40.27	177.5	0.91
1 a 4-gramy	197 148	40.91	294	0.27
1 a 5-gramy	197 177	40.92	303	0.26
1-gramy a slova	192 538	39.96	349	1.22
1,2 a 3-gramy	197 197	40.92	314.5	0.26
1,2 a 5-gramy	193 956	40.25	303	0.93
1,2-gramy a slova	190 808	39.60	329.5	1.58

Tabulka 19: Výsledky testování - plrabn12.txt



Obrázek 29: Dosažené kompresní poměry - plrabn12.txt

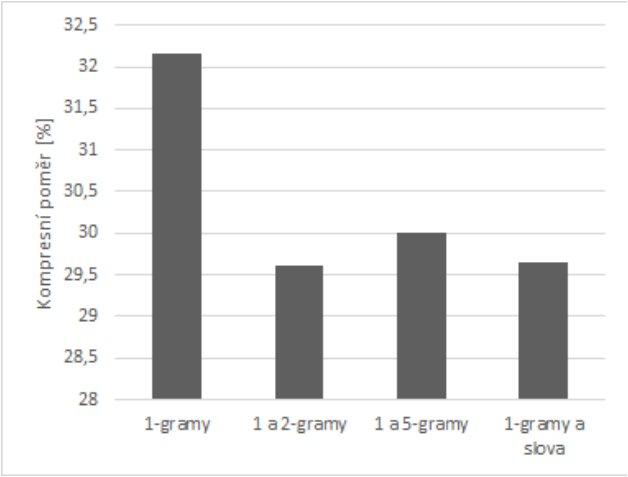
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 3.0, Step: 0.66, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.11 world192.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	2 473 400	x	x	x
1-gramy	795 192	32.15	94	x
1 a 2-gramy	732 073	29.60	354	2.55
1-gramy a slova	733 009	29.64	255	2.51
1 a 5-gramy	742 093	30.00	333	2.15

Tabulka 20: Výsledky testování - world192.txt



Obrázek 30: Dosažené kompresní poměry - world192.txt

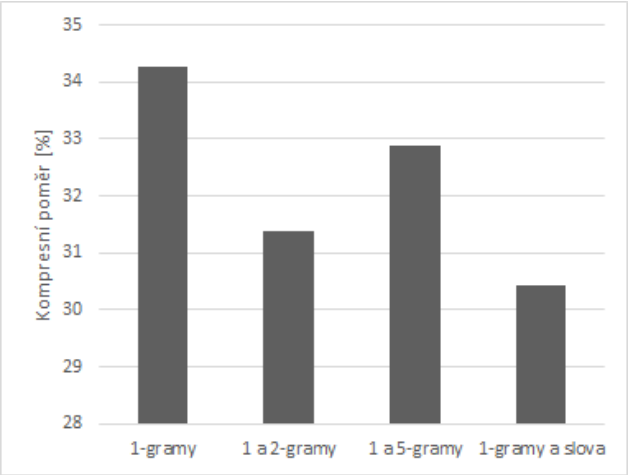
Všechny testy byly provedeny s konfigurací SOMA parametrů:

```
Mass: 1.5, Step: 0.74, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

10.3.12 bible.txt

	Velikost [B]	Kompresní poměr [%]	Použité symboly	Rozdíl 1-g a ev.abeceda
Originál	4 047 392	x	x	x
1-gramy	1 386 479	34.26	82	x
1 a 2-gramy	1 270 345	31.39	328	2.87
1-gramy a slova	1 231 809	30.43	241	3.83
1 a 5-gramy	1 330 570	32.87	342	1.39

Tabulka 21: Výsledky testování - bible.txt



Obrázek 31: Dosažené kompresní poměry - bible.txt

Testy byly provedeny s dvěma konfiguracemi SOMA parametrů:

```
Mass: 1.5, Step: 0.74, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

```
Mass: 3.0, Step: 0.99, PRT: 0.1, NP: 15, Migrations: 25, AcceptedError: -0.1
```

11 Závěr

Hlavním cílem této práce bylo vytvořit algoritmus, pro hledání optimální abecedy pro následnou kompresi čistého textu. Pro potřeby hledání bylo využito optimalizačního algoritmu SOMA. Kompresi dat obstarává existující metoda LZW.

Na základě výsledků závěrečného testování soudím, že se mi danou problematiku podařilo vyřešit a realizovat tak algoritmus, poskytující kvalitní řešení.

Výsledky testů ukazují, že u většiny testovaných souborů je užitečné přidat k 1-gramově založené abecedě především slova. Jejich přítomnost efektivně zvětšovala kompresní poměr u menších i u větších souborů. Největší rozdíl u této abecedy oproti znakové byl 5.37 % u souboru se zdrojovým kódem jazyka C. U souborů z Artificial corpus sekce je úroveň stlačení dat velká, jelikož se zde často opakují malé počty symbolů. U Canterbury corpus sekce se rozdíl hodnot evoluční abecedy a znakové pohybuje ve většině případů do 3 %. U souborů zdrojových kódů různých programovacích jazyků však hodnoty dosahují k 4 procentům. Large corpus testování neproběhlo v celém rozsahu. Důvodem je obrovská výpočetní, a tím i časová náročnost běhu algoritmu. Výsledné rozdíly se tady pohybují v průměru okolo 2,5 %.

Osobní přínos z tvorby této práce je nesporně v poznání principů optimalizačních algoritmů. Zvláště pak algoritmů z třídy evolučních, které se v poslední době hojně používají. Autor má nyní možnost při setkání se s optimalizačním problémem podat návrhy různých metod pro jeho řešení. Z toho také plyne schopnost programové implementace.

Přínos této práce obecně je v dalším rozvoji zkoumání komprese dat textového, i jiného charakteru, což je v současné době velmi aktuální téma.

Jako další možné vylepšení stávajícího případu vidím v použití více kombinací druhů symbolů a například jejich konkretizace pro určitý typ dat(odborné texty, zdrojové kódy, poezie).

Petr Němec

12 Reference

[1] Zelinka, Ivan, *Umělá inteligence v problémech globální optimalizace,* BEN-Technická literatura, 2002.

[2] Salomon, David, *Data Compression: The Complete Reference, Fourth Edition,* Springer, 2007.

[3] Weise, Thomas, *Global Optimization Algorithms – Theory and Application, e-book,* URL: http://www.it-weise.de, 2009. [cit. 10.4.2013]

[4] Bouchala, Jiří, *MATEMATIKA III,* URL: http://www.am.vsb.cz/bouchala, 2000. [cit. 22.3.2013]

[5] Tvrdík, Josef, *Evoluční algoritmy, Učební texty Ostravské Univerzity,* URL: http://www.prf.osu.cz, 2004. [cit. 25.3.2013]

[6] Kvasnička, V., Pospíchal, J., Tiňo, P., *Evolučné algoritmy,* STU Bratislava, 2000.

[7] Zelinka, I., Oplatková, Z., Šeda, M., Ošmera, P., Včelař, F., *Evoluční výpočetní techniky, Principy a aplikace,* BEN-Technická literatura, 2009.

[8] Coelho, L. dos Santos., *Self-organizing migration algorithm applied to machining allocation of clutch assembly, Mathematics and Computers in Simulation 80, str. 427-435,* Elsevier, 2009.

[9] Nollea L., Zelinka I., Hopgooda A.A., Goodyear A., *Comparison of an self-organizing migration algorithm with simulated annealing and differential evolution for automated waveform tuning, Advances in Engineering Software 36, 645–653,* Elsevier, 2005.

[10] Zelinka I., *Biologicky inspirované výpočty: Testovací funkce* URL: http://www.ivanzelinka.eu/hp/BIV.html [cit. 15.4.2013]

[11] Večerka, A., *Komprese dat,* Univerzita Palackého, Olomouc, URL: http://phoenix.inf.upol.cz, 2008. [cit. 18.4.2013]

[12] Lansky J., Zemlicka M., *Text compression: Syllables, vol. 129, str. 32-45,* DATESO, 2005.

[13] Krause, E. F., *Taxicab Geometry,* Dover, 1987.

[14] Wegner, P., *A technique for counting ones in a binary computer,* Communications of the ACM 3 (5), str. 322, 1960.